On the Existence of Stable Models of Non-stratified Logic Programs

Stefania COSTANTINI

Dip. di Informatica, Università di L'Aquila
via Vetoio Loc. Coppito, L'Aquila, I-67010 Italy
(*e-mail:* stefcost@di.univaq.it)

Abstract

In this paper we analyze the relationship between cyclic definitions and consistency in Gelfond-Lifschitz's answer sets semantics (initially defined as 'stable model semantics'). This paper introduces a fundamental result, which is very relevant for Answer Set programming, and planning. For the first time since the definition of the stable model semantics, the class of logic programs for which a stable model exists is given a syntactic characterization. This condition may have a practical importance both for defining new algorithms for checking consistency and computing answer sets, and for improving the existing systems. The approach of this paper is to introduce a new *canonical form* (to which any logic program can be reduced to), to focus the attention on cyclic dependencies. The technical result is then given in terms of programs in canonical form (canonical programs), without loss of generality: the stable models of any general logic program coincide (up to the language) to those of the corresponding canonical program. The result is based on identifying the cycles contained in the program, showing that stable models of the overall program are composed of stable models of suitable sub-programs, corresponding to the cycles, and on defining the *cycle graph.* Each vertex of this graph corresponds to one cycle, and each edge corresponds to one *handle,* which is a literal containing an atom that, occurring in both cycles, actually determines a connection between them. In fact, the truth value of the handle in the cycle where it appears as the head of a rule, influences the truth value of the atoms of the cycle(s) where it occurs in the body. We can therefore introduce the concept of a *handle path,* connecting different cycles. Cycles can be even, if they consist of an even number of rules, or vice versa they can be odd. Problems for consistency, as it is well-known, originate in the odd cycles. If for every odd cycle we can find a handle path with certain properties, then the existence of stable model is guaranteed. We will show that based on this results new classes of consistent programs can be defined, and that cycles and cycle graphs can be generalized to components and *component graphs.*

1 Introduction

In this paper we analyze the relationship between cyclic definitions and consistency in Gelfond-Lifschitz's answer sets semantics. As it is well-known, under the answer set semantics a theory may have no answer sets, since the corresponding general logic program may have no stable models (GelLif88) (GelLif91).

This paper introduces a fundamental result, which is relevant for Answer Set Programming (MarTru99), (Nie99) and planning (Lif99). For the first time, the class of logic programs for which a stable model exists is given a syntactic characterization (the result extends naturally to answer sets semantics) by providing a necessary and sufficient condition.

While checking for the existence of stable models is as hard a computational problem (in fact, NP-complete) as planning under certain assumptions (see (Lib99)), consistency checking is a good conceptual tool when derivations are based on consistency arguments. This is the case for approaches to planning that treat goals as constraints over models of the program. In general, constraints of the form: $\leftarrow g$ or, equivalently $f \leftarrow not\ f, g$ (where f does not appear elsewhere in the program) raise the expressivity of the language, though the programs where they appear are non-stratified. In Answer Set Planning, they become essential for expressing goals and conditions over goals. Then, being able to check for the existence of stable models syntactically for every answer set program can be of help for the logic programming encodings of planning (like, e.g., those of (Erd99), (FabLeoPfe99), (BBLMP00) and (DNK97)), to guarantee that a plan achieving the goal exists.

The approach of this paper is to introduce a new *canonical form* to which any logic program can be reduced to. The technical result is then given in terms of programs in canonical form (canonical programs), without loss of generality. Canonical programs focus the attention on cyclic dependencies. Rules are kept short, so as to make the syntactic analysis of the program easier. The stable models of any general logic program coincide (up to the language) to those of the corresponding canonical program.

A detailed analysis of the cost involved in reducing programs to their canonical form has been performed in (CosPro03) and, as intuition suggests, this transformation is tractable. Nevertheless, all definitions and results presented in this paper might be rephrased for general programs without conceptual problems, just at the expense of a lot of additional details. This means that reduction to canonical form is not strictly required neither for the theory, nor for an implementation.

The main result of this paper is a necessary and sufficient syntactic condition for the existence of stable models. On the one hand, this condition is of theoretical interest, as it is the first one ever defined since the introduction of the stable model semantics in (GelLif88). On the other hand, it may have a practical importance both for defining new algorithms for checking consistency and computing answer sets, and for improving the existing systems (Systems).

The result is based on identifying the cycles contained in the program, showing that stable models of the overall program are composed of stable models of suitable sub-programs, corresponding to the cycles, and on representing the program by means of the *cycle graph.* Each vertex of this graph corresponds to one cycle, and each edge corresponds to one *handle,* which is a literal containing an atom that, occurring in both cycles, actually determines

a connection between them. In fact, the truth value of the handle in the cycle where it appears as the head of a rule, influences the truth value of the atoms of the cycle(s) where it occurs in the body. We can therefore introduce the concept of a *handle path,* connecting different cycles. Cycles can be even, if they consist of an even number of rules, or vice versa they can be odd. Problems for consistency, as it is well-known, originate in the odd cycles. If for every odd cycle we can find a handle path with certain properties, then the existence of stable model is guaranteed.

The necessary and sufficient condition that we introduce is syntactic in the sense that it does not refers either to models or derivations. Checking this condition does not require to find the stable models, or to apply the rules of the program. It just requires to look at the program (represented by the cycle graph) and at the rules composing the cycles. The condition can however be exploited, so as to obtain: (i) new algorithms for finding the stable models, which are at least of theoretical interest; (ii) a new method for consistency checking divided into two steps: a first step related to the coarse structure of the program, that can be easily checked on the cycle graph so as to rule out a lot of inconsistent programs, thus leaving only the potentially consistent ones to be verified in a second step, that can be performed according to the approach presented here, or in any other way.

We will argue that the approach can also be useful for defining classes of programs that are consistent by construction, and as a first step toward a component-based methodology for the construction and analysis of answer set programs. This by means of a further generalization of cycle graphs to *component graphs,* where vertices are *components* consisting of bunches of cycles, and edges connect different components. We will argue that, in this framework, components can even be understood as independent agents.

It is useful to notice that in Answer Set Programming graph representations have been widely used for studying and characterizing properties of answer set programs, first of all consistency, and for computing the answer sets. Among the most important approaches we may mention the Rule Graph (DimTor96) and its extensions (Lin01) (Lin03b) (KonSchLin03a), and the Extended Dependency Graph (BCDP99), that we have considered and compared (Cos01), (CosDanPro02). Enhanced classes of graphs have been recently introduced in order to cope with extensions to the basic paradigm such as for instance preferences (KonSchLin03b) or nested logic programs (Lin03a). However, the cycle graph proposed in this paper is different form all the above-mentioned approaches, since its vertices are not atoms or rules, but significant subprograms (namely cycles), and the edges are connections between these subprograms.

2 Cycles and Handles

Assume the standard definitions of (propositional) general logic program and of Well-founded (VanGRosSch90) and stable models semantics (GelLif88). Whenever we mention consistency (or stability) conditions, we refer to the conditions introduced in (GelLif88). Let Π be a general logic program. In the following, we will often simply say "logic program" to mean a general logic program. Let $WFS(\Pi)$ be the well-founded model of Π.

Definition 1

A program Π is WF-irreducible if and only if $WFS(\Pi) = \langle \emptyset, \emptyset \rangle$.

That is, in a WF-irreducible programs all the atoms involved have truth value *undefined* under the Well-founded semantics.

For general logic programs, atoms with truth value "undefined" under the Well-founded semantics are exactly the atoms which are of interest for finding the stable models. This is a consequence of the fact that all stable models of a program *extend* its Well-founded model (VanGRosSch90), i.e., every literal which is true (resp. false) in the Well-founded model is also true (resp. false) in all stable models. Since (Cos95) we have proposed and refined over time a methodology for finding the stable models based on simplifying the given program w.r.t. the Well-founded semantics, thus obtaining a WF-irreducible reduct. The stable models of the original program can be easily obtained from the stable models of the reduct. Vice versa, if the reduct has no stable models the same holds for the original program.

For instance, for program

$p \leftarrow$ *not* p, *not* q

with Well-founded model $\langle \emptyset; \{q\} \rangle$ where atom p has truth value "undefined", we get the reduct $p \leftarrow$ *not* p by getting rid of a literal which is true under the Well-founded semantics, and thus is true in all stable models (if any exists). The reduct has no stable models, like the original program.

The following definitions are aimed at introducing a canonical form for any given WF-irreducible program. In the rest of this paper, we rely on the assumption that the order of literals in the body of rules is irrelevant.

Definition 2

A set of rules C is called a cycle if it has the following form:

$\lambda_1 \leftarrow$ *not* λ_2, Δ_1
$\lambda_2 \leftarrow$ *not* λ_3, Δ_2
...
$\lambda_n \leftarrow$ *not* λ_1, Δ_n

where $\lambda_1, \ldots, \lambda_n$ are distinct atoms. Each Δ_i, $i \leq n$, is a (possibly empty) conjunction $\delta_{i_1}, \ldots, \delta_{i_h}$ of literals (either positive or negative), where for each δ_{i_j}, $i_j \leq i_h$, $\delta_{i_j} \neq \lambda_i$ and $\delta_{i_j} \neq$ *not* λ_i. The Δ_i's are called the AND handles of the cycle. We say that Δ_i is an AND handle for atom λ_i, or, equivalently, an AND handle referring to λ_i.

We say that C has size n and it is even (respectively odd) if $n = 2k$, $k \geq 1$ (respectively, $n = 2k + 1$, $k \geq 0$). By abuse of notation, for $n = 1$ we have the (odd) self-loop $\lambda_1 \leftarrow$ *not* λ_1, Δ_1. In what follows, again by abuse of notation λ_{i+1} will denote $\lambda_{(i+1) mod n}$, i.e., $\lambda_{n+1} = \lambda_1$.

Given cycle C, we call $Composing\_atoms(C) = \{\lambda_1, \ldots, \lambda_n\}$ the set containing all the atoms *involved* in cycle C. We say that the rules listed above are *involved* in the cycle, or *form* the cycle. In the rest of the paper, sometimes it will be useful to see

$Composing\_atoms(C)$ as divided into two subsets, that we indicate as two *kinds* of atoms: the set of the *Even_atoms(C)* composed of the λ_i's with i even, and the set *Odd_atoms(C)*, composed of the λ_i's with i odd.

Conventionally, in the rest of the paper with C or C_i we will refer to cycles in general, with OC or OC_i to odd cycles and with EC or EC_i to even cycles

In the following program for instance, there is an odd cycle, that we may call OC_1, with composing atoms $\{e, f, g\}$ and an even cycle, that we may call EC_1, with composing atoms $\{a, b\}$.

— OC_1
$e \leftarrow$ *not* f, *not* a
$f \leftarrow$ *not* g, b
$g \leftarrow$ *not* e
— EC_1
$a \leftarrow$ *not* b
$b \leftarrow$ *not* a

OC_1 has an AND handle *not* a referring to e, and an AND handle b referring to f.

Notice that the sets of atoms composing different cycles are not required to be disjoint. In fact, the same atom may be involved in more than one cycle. The set of atoms composing a cycle can even be a proper subset of the set of atoms composing another cycle, like in the following program, where there is an even cycle EC_1 with composing atoms $\{a, b\}$, since a depends on *not* b and b depends on *not* a, but also and an odd cycle OC_1 with composing atom $\{a\}$, since a depends on *not* a.

— EC_1
— OC_1
$a \leftarrow$ *not* a, *not* b
$b \leftarrow$ *not* a

Here, OC_1 has an AND handle *not* b referring to a, but, symmetrically, EC_1 has an AND handle *not* a referring to b.

Thus, it may be the case that a handle of a cycle C contains an atom α which is involved in C itself, because α is also involved in some other cycle C_1.

Definition 3

A rule is called an *auxiliary rule of cycle* C (or, equivalently, *to* cycle C) if it is of this form:

$\lambda_i \leftarrow \Delta$

where $\lambda_i \in$ Composing_Atoms(C), and Δ is a non-empty conjunction $\delta_{i_1}, \ldots, \delta_{i_h}$ of literals (either positive or negative), where for each δ_{i_j}, $i_j \leq i_h$, $\delta_{i_j} \neq \lambda_i$ and $\delta_{i_j} \neq$ *not* λ_i. Δ is called an OR handle of cycle C (more specifically, an OR handle for λ_i or, equivalently, and OR handle referring to λ_i).

A cycle may possibly have several auxiliary rules, corresponding to different OR handles. In the rest of this paper, we will call *Aux(C)* the set of the auxiliary rules of a cycle C.

In the following program for instance, there is an odd cycle OC_1 with composing atoms $\{c, d, e\}$ and an even cycle EC_1 with composing atoms $\{a, b\}$. The odd cycle has three auxiliary rules.

— OC_1
$c \leftarrow$ *not* d
$d \leftarrow$ *not* e
$e \leftarrow$ *not* c
—— $Aux(OC_1)$
$c \leftarrow$ *not* a
$d \leftarrow$ *not* a
$d \leftarrow$ *not* b
— EC_1
$a \leftarrow$ *not* b
$b \leftarrow$ *not* a

In particular, we have $Aux(OC_1) = \{c \leftarrow$ *not* $a, d \leftarrow$ *not* $a, d \leftarrow$ *not* $b\}$.

In summary, a cycle may have some AND handles, occurring in one or more of the rules that form the cycle itself, and also some OR handles, occurring in its auxiliary rules.

All possible situations are enumerated, exemplified and discussed in (CosPro03). Cycles and handles can be unambiguously identified on the Extended Dependency Graph (EDG) of the program (BCDP99).

A cycle may also have no AND handles and no OR handles, i.e., no handles at all, in which case it is called *unconstrained.*The following program is composed of unconstrained cycles (in particular, there is an even cycle involving atoms a and b, and an odd cycle involving atom p).

— EC_1
$a \leftarrow$ *not* b
$b \leftarrow$ *not* a
— OC_1
$p \leftarrow$ *not* p

3 Canonical programs

In order to analyze the relationship between cycles, handles and consistency, below we introduce a *canonical form* for logic programs. This canonical form is new, and has been proposed and discussed in the companion paper (CosPro03) with the general objective of simplifying the study of formal properties of logic programs under the Answer Set semantics. Rules in *canonical programs* are in a standard format, so as to make definitions and proofs cleaner and easier to read. There is however no loss of generality, since, as proved in (CosPro03), any logic program can be *reduced* to a canonical program, and that stable models of the canonical program coincide (up to the language) with the stable models of the original program. In (CosPro03), an algorithm is described for obtaining the canonical form of a logic program, and the complexity analysis of the transformation is performed.

Definition 4
A logic program Π is in canonical form (or, equivalently, Π is a canonical program) if it is WF-irreducible, and fulfills the following syntactic conditions.

1. every atom in Π occurs both in the head and in the body of some rule;
2. every atom in Π is involved in some cycle;
3. each rule of Π is either involved in a cycle, or is an auxiliary rule of some cycle;
4. each handle of a cycle C consists of exactly one literal, either α or *not* α, where atom α does not occur in C.

Since the above definition requires handles to consist of just one literal, it implies that in a canonical program Π : (i) the body of each rule which is involved in a cycle consists of either one or two literals; (ii) the body of each rule which is an auxiliary rule of some cycle consists of exactly one literal.

Nothing prevents a rule to be at the same time involved in a cycle, and auxiliary to some other cycle. In this case however, the definition requires the rule to have exactly one literal in the body, i.e., the rule cannot have an AND handle.

As described in detail in (CosPro03), the transformation of a program into its canonical form consists in:

- Simplifying the program with respect to it Well-Founded model;
- Eliminating the "top" rules, i.e., those rules whose head atom does not occur in the body of any other rule;
- transforming long rules into new cycles (this by possibly adding new atoms), or into new rules that extend existing cycles;
- eliminating "bridges", i.e., intermediate steps in cycles or between cycles.

Consider for instance the program

$a \leftarrow$ *not* b, c, d
$b \leftarrow$ *not* a, e
$e \leftarrow b$
$d \leftarrow$ *not* a
$c \leftarrow$ *not* b

Its Well-Founded model is $\langle\{b, e\}\{c\}\rangle$ where atoms a and d are undefined. By simplifying with respect to this WFS (Cos95), we cancel literals which are true in the WFS and rules containing literals which are false in the WFS. We get:

$a \leftarrow d$
$d \leftarrow$ *not* a

that can be reduced by a further step (eliminating the "top", i.e., the first rule) to the canonical program

$a \leftarrow$ *not* a

Since the canonical program is inconsistent, so is the original program.

In fact (CosPro03), a program is consistent if and only if its canonical counterpart is consistent, and answer sets of the original program can be obtained in linear time from answer sets of the canonical program.

All the following definitions and proofs might be rephrased for the general case, but the choice of referring to canonical programs is a significant conceptual simplification that leads without loss of generality to a more readable and intuitive formalization. Notice for instance that in canonical programs the problem of consistency arises only in cycles containing an odd number of rules, since cycles do not have non-negated composing atoms.

Although for a detailed discussion we refer to (CosPro03), it is important to recall that canonical programs are WF-irreducible. For instance, the program

$p \leftarrow$ *not* p, q
$q \leftarrow$ *not* q, p

may look canonical, while it is not, since it has a non-empty Well-Founded model $\langle \emptyset; \{p, q\} \rangle$. In particular, since there are no undefined atoms, the set of true atoms of the Well-Founded model (in this case $\emptyset$) coincides (as it is well-known) with the unique stable model.

Similarly, the program

$q \leftarrow$ *not* q
$q \leftarrow p$

may look canonical, while it is not, since it has a non-empty Well-Founded model $\langle \emptyset; \{p\} \rangle$. Atom q is undefined. The corresponding canonical program is $q \leftarrow$ *not* q that, like the original program, has no stable models. The second rule is dropped by canonization, since its condition is false w.r.t. the WFM.

The program

$q \leftarrow p$
$p \leftarrow$ *not* r
$r \leftarrow$ *not* q

is not canonical because atom p is not involved in any cycle. In fact, in order to be involved in a cycle an atom must occur in the head of some rule but, also, its negation must occur in the body of some other rule. Here, atom p forms an (inessential) intermediate step between the two atoms q and r that actually form a cycle. The canonical form of the program is $q \leftarrow$ *not* $r, r \leftarrow$ *not* q, and puts the cycle into evidence. Given the stable models $\{q\}$ and $\{r\}$ of the canonical program, the stable models $\{p, q\}$ and $\{r\}$ of the original program can be easily obtained, since the truth value of p directly depends on that of r.

In the following, let the program at hand be a logic program Π in canonical form. Let $C_1, \ldots, C_w$ be all the cycles occurring in Π (called *the composing cycles* of Π). Whenever we will refer to C, C_1, C_2 etc. we implicitly assume that these are cycles occurring in Π.

4 Active handles

In this Section, we make some preliminary steps toward establishing a connection between syntax (cycles and handles) and semantics (consistency of the program). Truth or falsity of the atoms occurring in the handles of a cycle w.r.t. a given interpretation affects truth/falsity of the atoms involved in the cycle. As we discuss at length in the rest of the paper, this creates the conditions for stable models to exist or not.

Handles may help avoid inconsistencies in two ways. An AND handle Δ_i which is false allows the head λ_i of the rule to be false. An OR handle Δ which is true forces the atom λ_i to which it refers to be true as well. This idea is formalized into the following definitions of *active handles,* that will be widely used in the rest of the paper.

Definition 5

Let $\mathcal{I}$ be an interpretation. An AND handle Δ of cycle C is active w.r.t. $\mathcal{I}$ if the corresponding literal is false w.r.t. $\mathcal{I}$. We say that the rule where the handle occurs has an active AND handle. An OR handle Δ of cycle C is active w.r.t. $\mathcal{I}$ if the corresponding literal is true w.r.t. $\mathcal{I}$. We say that the rule where the handle occurs has an active OR handle.

Assume that $\mathcal{I}$ is a model. We can make the following observations. (i) The head λ of a rule ρ with an active AND handle is not required to be true in $\mathcal{I}$. (ii) The head of a rule $\lambda \leftarrow \Delta$ where Δ is an active OR handle is necessarily true in $\mathcal{I}$: since the body is true, the head λ must also be true.

Observing which are the active handles of a cycle C gives relevant indications about whether an interpretation $\mathcal{I}$ is a stable model.

Consider for instance the following program:

— OC_1
$p \leftarrow$ *not* p, *not* a
— EC_1
$a \leftarrow$ *not* b
$b \leftarrow$ *not* a
— OC_2
$q \leftarrow$ *not* q
— $Aux(OC_2)$
$q \leftarrow f$
— EC_2
$e \leftarrow$ *not* f
$f \leftarrow$ *not* e

Interpretations $\{a, f, q\}$, $\{a, e, q\}$, $\{b, p, f, q\}$ $\{b, p, e, q\}$ are minimal models. Consider interpretation $\{a, f, q\}$: both the AND handle *not* a of cycle $p \leftarrow$ *not* p, *not* a and the OR handle f of cycle $q \leftarrow$ *not* q are active w.r.t. this interpretation. $\{a, f, q\}$ is a stable model, since atom p is forced to be false, and atom q is forced to be true, thus avoiding the inconsistencies. In all the other minimal models instead, one of the handles is not active. I.e., either literal *not* a is true, and thus irrelevant in the context of a rule which is inconsistent, or literal f is false, thus leaving the inconsistency on q. These minimal models are in fact not stable.

In conclusion, the example suggests that for a minimal model $\mathcal{M}$ to be stable, each odd cycle must have an active handle. This will be stated formally in the next section.

Another thing that the example above shows is that the stable model $\{a, f, q\}$ of the overall program is actually the union of the stable model $\{a\}$ of the program fragment $OC_1 \cup EC_1$ and of the stable model $\{f, q\}$ of the program fragment $OC_2 \cup Aux(OC_2) \cup EC_2$. This is not by chance, and in the next Sections we will study how to relate the

existence of stable models of the overall program to the existence of stable models of the composing cycles. This relation is far from obvious, as demonstrated by the following simple program.

— OC_1
$p \leftarrow$ *not* p, *not* a
— EC_1
$a \leftarrow$ *not* b
$b \leftarrow$ *not* a
— OC_2
$q \leftarrow$ *not* q, *not* b

In this case, we have only one even cycle, and we might consider the program fragments: (i) $OC_1 \cup EC_1$ with stable model $\{a\}$, based on the active handle *not* a; (ii) (i) $OC_1 \cup EC_1$ with stable model $\{b\}$, based on the active handle *not* b. Unfortunately, the union $\{a, b\}$ of the stable models of the subparts is a minimal model but is not stable. In fact, neither atom a nor atom b is supported. This because the unconstrained even cycle EC_1, taken per se, has stable models $\{a\}$ and $\{b\}$, which are alternative and cannot be merged: this cycle in fact states that a holds if b does not hold, and vice versa. Thus, EC_1 cannot provide active handles for both the odd cycles.

Then, if we want to check whether a minimal model is stable, we not only have to check that every odd cycle has an active handle w.r.t. that model, but also that these handles do not enforce contradictory requirements on the even cycles. Similarly, we can try to build a stable model of the overall program out of the stable models of the composing cycles, taking however care of avoiding inconsistencies on the handles.

In summary, necessary and sufficient conditions for the existence of stable models can be obtained (and useful sufficient conditions can be derived from them) by taking the composing cycles, finding their stable models, and check if the corresponding active handles are in accordance. In order to do so, some preliminary definitions are in order.

It is useful to collect the set of handles of a cycle into a set, where however each handle is annotated so as to keep track of its *kind.* I.e., we want to remember whether a handle is an OR handle or an AND handle of the cycle.

Definition 6
Given cycle C, the set H_C of the handles of C is defined as follows, where $\beta \in Composing\_Atoms(C)$:

$$H_C = \{(\Delta : AND : \beta) \mid \Delta \text{ is an AND handle of } C \text{ referring to } \beta\} \cup \{(\Delta : OR : \beta) \mid \Delta \text{ is an OR handle of } C \text{ referring to } \beta\}$$

Whenever we need not care about β we shorten $(\Delta : K : \beta)$ as $(\Delta : K)$, K = AND/OR. By abuse of notation, we call "handles" the expressions in both forms, and whenever necessary we implicitly shift from one form to the other one. Informally, we will say for instance "the OR (resp. AND) handle Δ of β" meaning $(\Delta : OR : \beta)$ (resp. $(\Delta : AND : \beta)$).

The definitions below for instance does not rely on β.

Definition 7
The handles $(\Delta : AND)$ and $(\Delta : OR)$ are called *opposite handles.* Given a handle h, we will indicate its opposite handle with h^-.

Definition 8
The handles $(\Delta_1 : K)$ and $(\Delta_2 : K)$ are called *contrary handles* if $\Delta_1 = \alpha$ and $\Delta_2 =$ *not* α. Given a handle h, we will indicate its contrary handle with h^n.

Whenever either contrary or opposite couples of handles occur in a program, even for different β's, if one is active the other one is not active, and vice versa.

Definition 9
The handles $(\Delta_1 : K1)$ and $(\Delta_2 : K2)$ are called *sibling handles* if $K1 \neq K2$, and $\Delta_1 = \alpha$ and $\Delta_2 =$ *not* α. Given a handle h, we will indicate its sibling handle with h^s.

Whenever sibling couples of handles occur in a program, even for different β's, if one is active the other one is active as well.

Taken for instance atom α, we have:

- $(\alpha : AND)$ and $(\alpha : OR)$ are opposite handles;
- $(not\ \alpha : AND)$ and $(not\ \alpha : OR)$ are opposite handles;
- $(\alpha : AND)$ and $(not\ \alpha : AND)$ are contrary handles;
- $(\alpha : OR)$ and $(not\ \alpha : OR)$ are contrary handles;
- $(\alpha : OR)$ and $(not\ \alpha : AND)$ are sibling handles;
- $(\alpha : AND)$ and $(not\ \alpha : OR)$ are sibling handles;

In general however the indication of β is necessary. In fact, different atoms of a cycle may have handles with the same Δ, but although active/not active at the same time, they may affect the existence of stable models differently. Take for instance the following program with the indication of the composing cycles:

— OC_1
$q \leftarrow not\ q, e$
$q \leftarrow not\ a$
— OC_2
$a \leftarrow not\ b, not\ e$
$b \leftarrow not\ c, not\ f$
$c \leftarrow not\ a, not\ e$
— OC_3
$p \leftarrow not\ p, not\ e$
— EC_1
$e \leftarrow not\ f$
$f \leftarrow not\ g$

we have $H_{OC_1} = \{(e : AND : q), (not\ a : OR : q)\}$, $H_{OC_2} = \{(not\ e : AND : a), (not\ f : AND : b), (not\ e : AND : c)\}$, $H_{OC_3} = \{(not\ e : AND : p)\}$, $H_{EC_1} = \emptyset$.
Handle $(not\ e : AND)$ occurs several times, even twice in cycle OC_2, referring to different atoms. While however OC_2 may in principle rely upon two different AND handles for

keeping consistency, OC_3 can rely only upon this one, which means that if it is not active the whole program is surely inconsistent.

Given any subset Z of H_C, it is useful to identify the set of atoms occurring in the handles belonging to Z.

Definition 10
Let $Z \subseteq H_C$. The set of the atoms occurring in the handles belonging to Z is defined as follows.

$$\begin{array}{rcl} Atoms(Z) & = & \{\alpha \,|\, (\alpha : K) \in Z\} \ \cup \\ & & \{\alpha \,|\, (\mathit{not}\ \alpha : K) \in Z\} \end{array}$$

If for instance we take $Z = H_{OC_1}$, we have $Atoms(H_{OC_1}) = \{a, e\}$.

Given any subset Z of H_C, it is useful to state which are the atoms that are required to be true, in order to make all the handles in Z active (implicitly, to this aim all the other atoms are required to be false).

Definition 11
Let $Z \subseteq H_C$. The set of atoms $ActivationAt_C(Z) \subseteq Atoms(Z)$ is defined as follows.

$$\begin{array}{rcl} ActivationAt_C(Z) & = & \{\alpha \,|\, (\alpha : OR) \in Z\} \qquad \cup \\ & & \{\alpha \,|\, (\mathit{not}\ \alpha : AND) \in Z\} \end{array}$$

If for instance we take $Z = H_{OC_1}$, we have $ActivationAt(H_{OC_1}) = \{e\}$.

Vice versa, any subset V of $Atoms(H_C)$ corresponds to a subset of the handles of C that become active, if atoms in V are true.

Definition 12
Let $V \subseteq Atoms(H_C)$.

$$\begin{array}{rcl} Active_C(V) & = & \{(\Delta : AND) \,|\, \Delta = \mathit{not}\ \alpha, \alpha \in V\} \ \cup \\ & & \{(\Delta : OR) \,|\, \Delta = \alpha, \alpha \in V\} \end{array}$$

If for instance we take $V = \{a, e\}$ for cycle OC_1, we have $Active_{OC_1}(\{a, f\}) = \{((e : AND))\}$.

Finally, it is useful to introduce a short notation for the union of different sets of rules.

Definition 13
Let $I_1, \ldots I_q$ be sets of rules. As a special case, some of the I_j's can be sets of atoms, where each atom $\beta \in I_j$ is understood as a fact $\beta \leftarrow$. By $I_1 + \ldots + I_q$ we mean the program consisting of the union of all the rules belonging to $I_1, \ldots I_q$. By $\mathbb{B}_{\mathrm{I_1,\ldots I_q}}$ we mean the Herbrand base of program $I_1 + I_2 \ldots + I_q$.

5 Active handles and existence of stable models

In this one and the following sections we proceed further toward a framework that relates cycles, handles and active handles to the existence of stable models. As a first result, we are able to prove the following:

Theorem 1

Let Π be a program, and let $\mathcal{M}$ be a minimal model of Π. $\mathcal{M}$ is a stable model only if each odd cycle OC_i occurring in Π has an active handle w.r.t. $\mathcal{M}$.

Proof

By (Cos95), Corollary 3.1., since there are no positive cycles in Π, minimal model $\mathcal{M}$ is stable if and only if for each $A \in \mathcal{M}$ there exists a rule in Π with head A, and body which is true w.r.t. $\mathcal{M}$, i.e., a rule which *supports* A. Let $x \ mod \ y$ be (as usual) the remainder of the integer division of x by y.

Assume that $\mathcal{M}$ is stable, but there is an odd cycle without active handles, composed of atoms $\lambda_1, \ldots, \lambda_n$, n odd. Take a λ_i, and assume first that $\lambda_i \in \mathcal{M}$. Since there is no active OR handle, each λ_i can possibly be supported only by the corresponding rule in the cycle. By definition of cycle, this rule has the form:

$$\lambda_i \leftarrow \mathit{not}\ \lambda_{(i+1)\ mod\ n}, \Delta_i$$

Since there are no active AND handles, then all Δ's are true w.r.t. $\mathcal{M}$. For λ_i to be supported, *not* $\lambda_{(i+1)\ mod\ n}$ should be true as well, i.e., $\lambda_{(i+1)\ mod\ n}$ should be false. The rule for $\lambda_{(i+1)\ mod\ n}$ has the form:

$$\lambda_{(i+1)\ mod\ n} \leftarrow \mathit{not}\ \lambda_{(i+2)\ mod\ n}, \Delta_{(i+1)\ mod\ n}$$

Since $\Delta_{(i+1)\ mod\ n}$ is true w.r.t. $\mathcal{M}$, for $\lambda_{(i+1)\ mod\ n}$ to be false, *not* $\lambda_{(i+2)\ mod\ n}$ should be false as well, i.e., $\lambda_{(i+2)\ mod\ n}$ should be true. By iterating this reasoning, $\lambda_{(i+3)\ mod\ n}$ should be false, etc. In general, $\lambda_{(i+k)\ mod\ n}$ should be false w.r.t. $\mathcal{M}$ with k odd, and true with k even. Then, since the number n of the composing atoms is odd, $\lambda_{(i+n)\ mod\ n}$ should be false w.r.t. $\mathcal{M}$, but $\lambda_{(i+n)\ mod\ n} = \lambda_i$, which is a contradiction. Assume now that $\lambda_i \notin \mathcal{M}$. Then, *not* λ_i is true w.r.t. $calm$, and thus, since the corresponding AND handle is not active, $\lambda_{(i-1)\ mod\ n}$ is supported and should belong to $\mathcal{M}$. Consequently, we should have $\lambda_{(i-2)\ mod\ n} \notin calm$. In general, $\lambda_{(i-k)\ mod\ n}$ should be true w.r.t. $\mathcal{M}$ with k odd, and false with k even. Then, since the number n of the composing atoms is odd, $\lambda_{(i-n)\ mod\ n}$ should be true w.r.t. $\mathcal{M}$, but $\lambda_{(i-n)\ mod\ n} = \lambda_i$, which is again a contradiction. □

Now, consider a cycle C_i together with its auxiliary rules, i.e., consider the set of rules $C_i + Aux(C_i)$ and take it as an independent program. There are atoms in this program that do not appear in the conclusion of rules: these are exactly the atoms occurring in the handles of C, i.e., the atoms in $Atoms(H_{C_i})$. Take a set $X_i \subseteq Atoms(H_{C_i})$. Assume to add atoms in X_i as facts to $C_i + Aux(C_i)$, thus obtaining $C_i + Aux(C_i) + X_i$, that we call *extended cycle* corresponding to X_i. With respect to the overall program, this addition *simulates* atoms in X_i to be concluded true in some other part of the program. Depending on the active handles corresponding to X_i, the extended cycle $C_i + Aux(C_i) + X_i$ may or may not be consistent.

Consider for instance the following program.

— OC_1
$q \leftarrow$ *not* q
—— $Aux(OC_1)$
$q \leftarrow f$
— OC_2
$p \leftarrow$ *not* p, *not* f
— EC_1
$e \leftarrow$ *not* f
$f \leftarrow$ *not* e

It can be seen as divided into the following parts, each one corresponding to $C_i + Aux(C_i)$ for some cycle C_i. The first part is composed of odd cycle OC_1, with an auxiliary rule (OR handle):

$q \leftarrow$ *not* q
$q \leftarrow f$

The second part is composed of odd cycle OC_2, without auxiliary rules but with an AND handle:

$p \leftarrow$ *not* p, *not* f

The third part is composed of the unconstrained even cycle EC_1:

$e \leftarrow$ *not* f
$f \leftarrow$ *not* e

OC_1 in itself is inconsistent, but if we take $X_{OC_1} = \{f\}$ we get an extended cycle with stable model $\{f, q\}$: the active OR handle forces q to be true. Similarly, for OC_2, if we take $X_{OC_2} = \{f\}$ we get an extended cycle with stable model $\{f\}$: the active AND handle forces p to be false. Cycle EC_1 is consistent, with stable models $\{e\}$ and $\{f\}$. If we now select the stable model $\{f\}$ for EC_1, we make X_{OC_1} and X_{OC_2} effective, thus obtaining the stable model $\{f, q\}$ for the overall program. Instead, the stable model $\{e\}$ for EC_1 does not serve to the purpose of obtaining a stable model for the overall program, since it does not make the handles of the odd cycles active, thus leaving the inconsistencies as they are.

Take now this very similar program, that can be divided into cycles analogously.

— OC_1
$q \leftarrow$ *not* q
—— $Aux(OC_1)$
$q \leftarrow f$
— OC_2
$p \leftarrow$ *not* p, *not* e
— EC_1
$e \leftarrow$ *not* f
$f \leftarrow$ *not* e

The difference is that OC_2 has AND handle *not* e (instead of *not* f). In order to make this handle active, we should take $X_{OC_2} = \{e\}$. In this case, the stable model $\{e\}$ for EC_1

should be selected, but unfortunately it does not suits OC_2, that still "requires" $\{f\}$. Then, no choice can be made for EC_1 so as to make the program consistent.

The above simple example explains what we will argue in the rest of the paper, and precisely that, for checking whether a logic program has stable models (and, possibly, for finding these models) one can do the following.

(i) Divide the programs into pieces, of the form $C_i + Aux(C_i)$, and check whether every odd cycle has handles; if not, then the program is inconsistent;
(ii) For every cycle C_i with handles, find the sets X_i that make the subprogram $C_i + Aux(C_i)$ consistent, and find the corresponding stable models S_{C_i}; notice that in the case of unconstrained even cycles, H_{C_i} is empty, and we have two stable models, namely $M^1_{C_i}$ = Even_atoms(C_i) and $M^2_{C_i}$ = Odd_atoms(C_i).
(iii) Check whether there exists a collection of X_i's, one for each cycle, such that the corresponding S_{C_i}'s agree on shared atoms: in this case the program is consistent, and its stable model(s) can be obtained as the union of the S_{C_i}'s.

In the ongoing we will show that we can check conditions (ii)-(iii) by representing the program by means of a kind of graph, whose vertices are the cycles and whose edges are marked with the handles. Formally:

Definition 14
Let C be a cycle. Let $Z \subseteq H_C$. The program $C + Aux(C) + Z$ is an *extended cycle* corresponding to C.

Definition 15
Let C_i be a cycle occurring in Π. We say that $S_{C_i} \subseteq \mathbb{B}_{\mathrm{C_i+Aux(C_i)}}$ is a *partial stable model* for Π relative to C_i, if $\exists X_i \subseteq Atoms(H_{C_i})$ such that S_{C_i} is a stable model of the corresponding extended cycle $C_i + Aux(C_i) + X_i$. The set X_i is called a positive base for S_{C_i}, while the set $X_i^- = Atoms(H_{C_i}) \setminus X_i$ is called a negative base for S_{C_i}. The couple of sets $\langle X_i, X_i^- \rangle$ is called a base for S_{C_i}.

As discussed above, atoms in X_i are added to simulate that we deduce them true in some other part of the program. Symmetrically, atoms in X_i^- are supposed not to be concluded true anywhere in the program. The positive base X_i may be empty: in this case, all the atoms occurring in the handles are supposed to be false. The choice of X_i corresponds to the choice of a specific set of active handles, namely $Active_{C_i}(X_i)$. There may be no partial stable models relative to a cycle C_i, or there may be different ones, depending on the possible choices of X_i's that correspond to consistent extended cycles.

By abuse of notation, when program Π and its composing cycles are uniquely identified in the context of the discussion, and C_i is a cycle occurring in Π, we will speak of partial stable models *of* C_i, meaning partial stable models for Π relative to C_i.

Once we get partial stable models of the composing cycles, we can try to put them together in order to obtain stable models for the whole program. Of course we will try to obtain each stable model of the overall program by taking one partial stable model for each cycle, and considering their union. This however will work only if the partial stable models assign truth values to atoms in a compatible way.

Definition 16

Consider a collection $\mathcal{S} = S_1, \ldots, S_w$ of partial stable models for Π, relative to its composing cycles $C_1, \ldots, C_w$, each S_i with base $\langle X_i, X_i{}^- \rangle$. We say that $S_1, \ldots, S_w$ are compatible or, equivalently, that $\mathcal{S}$ is a compatible set of partial stable models whenever the following conditions hold:

1. $\forall j, k \leq w$, $X_j \cap X_k{}^- = \emptyset$;
2. $\forall j \leq w, \forall A \in X_j, \exists h \neq j$ such that $A \in S_h$, and $A \notin X_h$;
3. $\forall j \leq w, \forall B \in X_j{}^-$, $\nexists t \leq w$ such that $B \in S_t$.

Condition (1) states that the bases of compatible partial stable models cannot assign opposite truth values to any atom. Condition (2) ensures that, if an atom A is supposed to be true in the base of some cycle C_j, it must be actually concluded true in some other cycle C_h. Notice that "concluded" does not mean "assumed", and thus A must occur in the partial stable model S_h of C_h, without being in its set of assumptions X_h. Condition (3) ensures that, if an atom is supposed to be false in the base of some cycle, it cannot be concluded true in any of the other cycles.

The following theorem formally states the connection between the stable models of Π, and the partial stable models of its cycles.

Theorem 2

An interpretation $\mathcal{I}$ of Π is a stable model if and only if there exists a compatible set $\mathcal{S} = S_1, \ldots, S_w$ of partial stable models for its composing cycles such that $I = \bigcup_{i \leq w} S_i$.

Proof

Suppose that $\mathcal{I}$ is a stable model for Π. Let C_i, $i \leq w$, be any of the composing cycles of Π. Let $X_i = \mathcal{I} \cap Atoms(H_{C_i})$, which means that X_i is the set of the atoms of the handles of C_i which are true w.r.t. $\mathcal{I}$. Let $S_i = \mathcal{I} \cap \mathbb{B}_{\mathrm{C_i + Aux(C_i)}}$ be the restriction of $\mathcal{I}$ to the atoms involved in the extended cycle corresponding to X_i. S_i is clearly a stable model for $P_i = C_i + Aux(C_i) + X_i$. In fact, all non-unit rules of P_i are also rules of Π. Therefore, would the consistency conditions be violated for P_i, they would be violated for Π as well.

Vice versa, let us consider a set $S = \{S_1 \cup \ldots \cup S_w\}$, of partial stable models for the cycles in Π. Notice that Π itself corresponds to the union of the cycles and of their auxiliary rules, i.e., $\Pi = \bigcup_{i \leq w} C_i + Aux(C_i)$.

Let us first show that S is a stable model of the program Π_L obtained as $\bigcup_{i \leq w} C_i + Aux(C_i) + X_i$, which is a superset of Π. In fact, each S_i satisfies the stability condition on the rules of the corresponding extended cycle, and, since they form a compatible set, by conditions (1) and (3) of Definition 16 no atom which is in the negative base of any of the S_i's, is concluded true in some other S_j. Therefore, S is a stable model of Π_L.

In order to obtain Π from Π_L, we have to remove the positive bases of cycles, which are the unit rules corresponding to the X_i's. By condition (2) however, in a set of compatible partial stable models, every atom $A \in X_i$ is concluded true in some S_j, $i \neq j$, i.e., in the partial stable model of some other cycle. This implies that S satisfies the stability condition also after X_i's have been removed: then, S is a stable model for Π. □

Each stable model S of Π corresponds to a different choice of of the X_i's, i.e., of the active handles of cycles.

The above result is of theoretical interest, since it sheds light on the connection between stable models of a program and stable models of its sub-parts. It may also contribute to any approach to modularity in software development under the stable model semantics. However, to the aim of developing effective software engineering tools and more efficient algorithms for computing stable models, syntactic conditions for the existence of stable models are in order. In the ongoing, we use this result as the basis for defining a necessary and sufficient syntactic condition for program consistency.

In this direction, consider the following corollaries:

Corollary 1
An odd cycle C occurring in Π has partial stable models if and only if it is constrained (i.e., it has at least one handle).

Whenever this handle is assumed to be active, the corresponding extended cycle has one partial stable model. We can thus state a necessary condition for the existence of stable models of Π.

Corollary 2
An interpretation S is a stable model of Π only if every odd cycle occurring in Π has at least one active handle w.r.t. S.

Otherwise in fact, any odd cycle without active handles would have no partial stable models. The above condition on the odd cycles is however not sufficient for the consistency of Π. This because the active handles of different odd cycles might correspond to inconsistent requirements on the truth values of the atoms involved. In order to find a necessary and sufficient condition, a finer investigation on the structure of program Π is needed.

6 Handle assignments and admissibility

An important observation that arises from the examples, and has been made formal in Theorem 2, is that in any stable model for Π different cycles cannot have active handles requiring opposite truth values of same atom α. A second observation is that the suitable truth value of atom α occurring in a handle must be derived in the cycles α is involved into, which are the cycles the handle *comes from,* or equivalently the *source cycles* of the handle.

Definition 17
A handle $(\Delta : K)$ of cycle C_1, $\Delta = \alpha$ or $\Delta =$ *not* α *comes from* source cycle C_2 if $\alpha \in Composing\_atoms(C_2)$.

Handles in H_C are called the *incoming handles* of C. The same handle of a cycle C may come from different cycles, and may refer to different atoms of C. For instance, in the program below we have:

— OC_1
$a \leftarrow$ *not* b, *not* f
$b \leftarrow$ *not* c
$c \leftarrow$ *not* a, *not* f
$b \leftarrow g$
— EC_1
$f \leftarrow$ *not* g
$g \leftarrow$ *not* f
— EC_2
$f \leftarrow$ *not* h
$h \leftarrow$ *not* f

handle $(not\ f\ :\ AND)$ of OC_1 comes from both EC_1 and EC_2, and refers to two different atoms in OC_1, namely a and c; handle $(g : OR)$ of OC_1 comes from EC_1, and refers to atoms b.

The following definition completes the coming-from terminology the other way round, by identifying the atoms occurring in handles coming from C.

Definition 18
Given cycle C, the set of the atoms involved in C that occur in the handles of some other cycle is defined as follows:
$Out\_handles(C) = \{\beta \mid$
$\beta \in Composing\_Atoms(C) \wedge \exists C_1$ such that $\beta \in Atoms(H_{C_1})\}$

In the above program for instance, $Out\_handles(EC_1) = \{f, g\}$ and $Out\_handles(EC_2) = \{f\}$.

For an handle to be active w.r.t. an interpretation, we must have the following. (i) If the corresponding atom α is required to be true, then it must be concluded true (by means of a supporting rule) in *at least one* of the cycles the handle comes from, which implies α to be concluded true in *all* the *extended* cycles it is involved into: in fact, the rule that makes α true is an auxiliary rule for all these cycles. (ii) If the corresponding atom α is required to be false, then it must be concluded false in *all* the (extended) cycles it comes from.

This is illustrated by the following example:

— OC_1
$p \leftarrow$ *not* p, *not* c
— OC_2
$c \leftarrow$ *not* d
$d \leftarrow$ *not* e
$e \leftarrow$ *not* c, f
— EC_1
$f \leftarrow$ *not* g
$g \leftarrow$ *not* f
— EC_2
$f \leftarrow$ *not* h
$h \leftarrow$ *not* f

The extended cycles are:

— OC_1
$p \leftarrow$ *not* p, *not* c
with no auxiliary rules, $Out\_handles(EC_1) = \emptyset$, $H_{OC_1} = \{$*not* $c \ : \ AND \ : \ p\}$, $Atoms(H_{OC_1}) = \{c\}$ and unique partial stable model $\{c\}$ obtained by choosing positive base $X_{OC_1} = \{c\}$;

— OC_2
$c \leftarrow$ *not* d
$d \leftarrow$ *not* e
$e \leftarrow$ *not* c, f
with no auxiliary rules, $Out\_handles(EC_1) = \{f\}$, $H_{OC_2} = \{f \ : \ AND \ : \ e\}$, $Atoms(H_{OC_2}) = \{f\}$ and unique partial stable model $\{d\}$ obtained by choosing positive base $X_{OC_2} = \emptyset$, $X^{-}_{OC_1} = \{f\}$;

— $EC_1 + Aux(EC_1)$
$f \leftarrow$ *not* g
$g \leftarrow$ *not* f
$f \leftarrow$ *not* h
with $Out\_handles(EC_1) = \{f\}$, $H_{EC_1} = \{$*not* $h \ : \ OR \ : \ f\}$, $Atoms(H_{EC_1}) = \{h\}$ and two partial stable models $\{f\}$ and $\{g\}$. The former one can be obtained either by choosing $X_{EC_1} = \emptyset$ or, also, $X_{EC_1} = \{h\}$. The latter one requires $X_{EC_1} = \{h\}$, so as to allow f to be false.

— $EC_2 + Aux(EC_2)$
$f \leftarrow$ *not* h
$h \leftarrow$ *not* f
$f \leftarrow$ *not* g
with $Out\_handles(EC_2) = \{f\}$, $H_{EC_2} = \{$*not* $g \ : \ OR \ : \ f\}$, $Atoms(H_{EC_2}) = \{g\}$ and two partial stable models $\{f\}$ and $\{h\}$. The former one can be obtained either by choosing $X_{EC_1} = \emptyset$ or, also, $X_{EC_1} = \{g\}$. The latter one requires $X_{EC_1} = \{g\}$, so as to allow f to be false.

Unfortunately, the overall program turns out to have no stable model, because: for obtaining the partial stable model of OC_2, f must be concluded false so as to make the unique AND handle active. Both EC_1 and EC_2 actually admit a partial stable model where f is false. Thus, for the fragment $EC_1 + EC_2 + OC_2$ we might construct the unique wider partial stable model $\{g, h, d\}$. However, this fails to make the handle of OC_1 active, and therefore a stable model for the overall program cannot be obtained.

If we replaced OC_1 with OC'_1 — OC_1
$p \leftarrow$ *not* p, *not* d

with $H_{OC'_1} = \{$*not* $d \ : \ AND \ : \ p\}$, $Atoms(H_{OC'_1}) = \{d\}$ and unique partial stable model $\{d\}$ obtained by choosing positive base $X_{OC_1} = \{d\}$, then $\{g, h, d\}$ we would be a stable model $\{\}$ for the overall program.

Below we establish the formal foundations of the kind of reasoning that we have informally proposed up to now. We introduce the definition of *handle assignment*, which is a

consistent hypothesis on (some of) the handles of a cycle C. Precisely, it is a quadruple composed of the following sets.

IN_C^A contains the incoming handles which are assumed to be active. From IN_C^A one can immediately derive a corresponding assumption on X_C. In particular, $X_C = ActivationAt_C(IN_C)$, i.e. it is exactly the set of the atoms that make the handles in IN_C^A active. Vice versa, IN_C^N contains the incoming handles which are assumed to be not active. Handles of C which are not in $IN_C^A \cup IN_C^N$ can be either active or not active, but their status is unknown or irrelevant in the context where the handle assignment is used.

OUT_C^+ is the set of out-handles which are required to be concluded true, so as to make some handle of some other cycle active (as we have seen in the example above). Similarly, OUT_C^- is the set of the out-handles which are required to be concluded false, for the same reason. Of course, the OUT_C's must be disjoint, since no atom can be required to be simultaneously true and false.

Definition 19

A basic handle assignment to (or for) cycle C is a quadruple of sets

$$\langle IN_C^A, IN_C^N, OUT_C^+, OUT_C^- \rangle$$

where the (possibly empty) composing sets are such that:
$IN_C^A \cup IN_C^N \subseteq H_C$;
$IN_C^A \cap IN_C^N = \emptyset$;
neither IN_C^A and IN_C^N contain couples of either opposite or contrary handles; $OUT_C^+ \cup OUT_C^- \subseteq Out\_handles(C)$;
$OUT_C^+ \cap OUT_C^- = \emptyset$.

A handle assignment will be called is *trivial* (resp. non-trivial) if $OUT_C^+ = OUT_C^- = \emptyset$, i.e., whenever there is no requirement on the out-handles of C.

If IN_C^A is empty, then either $H_C = \emptyset$, i.e., the cycle is unconstrained, or $H_C \neq \emptyset$ but no active incoming handle is assumed, in which case we say that the cycle is *actually unconstrained* w.r.t. this handle assignment. A handle assignment will be called *effective* whenever $IN_C^A \neq \emptyset$. For short, when talking of bot IN_C^A and IN_C^N we will say "the IN_C's".

We have to cope with the relationship between opposite, contrary, and sibling handles, whenever they should occur in the same cycle C.

Definition 20

A complete handle assignment, or simply a handle assignment, to cycle C is a basic handle assignment to C where the following conditions hold, whenever couples of opposite, contrary or sibling handles occur in C:
if two opposite handles h and h^- both occur in C, $h \in IN_C^A$ if and only if $h^- \in IN_C^N$;
if two contrary handles h and h^n both occur in C, $h \in IN_C^A$ if and only if $h^n \in IN_C^N$;
if two sibling handles h and h^s both occur in C, then either $h, h^s \in IN_C^A$ and $h, h^s \notin IN_C^N$ or $h, h^s \in IN_C^N$ and $h, h^s \notin IN_C^A$.

A basic handle assignment can be *completed*, i.e., turned into a complete handle assignment, by an obvious update of the IN_C's.

What the definition does not state yet is that IN_C's and the OUT_C's should be compatible, in the sense that the handles in IN_C^A and IN_C^N being active should not prevent the out-handles in OUT_C's from being true/false as required. Consider for instance the extended cycle:

— OC
$a \leftarrow$ *not* b, f
$b \leftarrow$ *not* c
$c \leftarrow$ *not* a
$b \leftarrow$ *not* e
where $H_{EC} = \{(e : OR : c), (f : AND : a)\}$.
Let us assume that $Out\_handles(OC) = \{a, b\}$.
Now take a handle assignment with the following components. $IN_{OC}^A = \{(f : AND : a)\}$ which means that we assume this handle to be active, i.e., we assume f to be false. $IN_{OC}^N = \{(e : OR : b)\}$, which means that we assume this handle to be not active, i.e., we assume f to be false. Finally, $OUT_{OC}^+ = \{b\}$, and $OUT_{OC}^- = \{c\}$. This handle assignment cannot be fulfilled in practice: in fact, if f is assumed to be false, then a is concluded false, and consequently c is concluded true and b false, contrary to what required in OUT_{OC}^+ and OUT_{OC}^-. The OR handle f of b is in IN_{OC}^N, and thus it is assumed to be not active. Notice that even with $IN_{OC}^N = \emptyset$, i.e., with no knowledge about handle $(e : OR : b)$, still with the information that we possess, the requirements for OUT_{OC}^+ and OUT_{OC}^- cannot be fulfilled. Instead, an handle assignment with the same IN_{OC}'s, and with $OUT_{OC}^+ = \{c\}$ and $OUT_{OC}^- = \emptyset$ can be fulfilled. Notice also that $OUT_{OC}^- = \emptyset$ does not mean that no out-handle is *allowed* to be false, rather it means that no out-handled is *required* to be false. Then, if the requirements in OUT_{OC}^+ and OUT_{OC}^- are met, the remaining out-handles can take any truth value. Notice finally that if we let $IN_{OC}^A = IN_{OC}^N = \emptyset$, then the extended cycle is inconsistent.

Clearly, a definition of the IN_{OC}'s that makes the corresponding program fragment $C + Aux(C) + ActivationAt_C(IN_C^A)$ inconsistent is useless for obtaining stable models of the overall program. In fact, we are interested in handle assignments where the IN_{OC}'s corresponds to an assumption on the incoming handles (and thus on $X_C = ActivationAt_C(IN_C^A)$) such that: the resulting program fragment $C + Aux(C) + X_C$ is consistent, and the requirements established in OUT_{OC}^+ and OUT_{OC}^- are met. This means that in some stable model of the program fragment, all atoms in OUT_{OC}^+ are deemed true, and all atoms in OUT_{OC}^- are deemed false.

The above requirements are formalized in the following:

Definition 21

A handle assignment $HA = \langle IN_C^A, IN_C^N, OUT_C^+, OUT_C^- \rangle$ to a cycle C is admissible if and only if the program $C + Aux(C) + ActivationAt_C(IN_C^A)$ is consistent, and for some stable model $S^{IN_C^A}$ of this program, $OUT_C^+ \subseteq S^{IN_C^A}$ and $OUT_C^- \cap S^{IN_C^A} = \emptyset$. We say that $S^{IN_C^A}$ corresponds to HA.

According to Definition 15, each stable model $S^{IN_C^A}$ is a partial stable model of Π relative to C, that can be used for building a stable model of the whole program. At least

some of these partial stable models *correspond* to the given handle assignment, in the sense that they are consistent with the choice of active handles that the assignment represents.

It is useful to notice that: (i) if the cycle C is even, and it is either unconstrained or actually unconstrained, then the program fragment $C + Aux(C) + ActivationAt_C(IN_C^A)$ has two stable models , one coinciding with $Even\_atoms(C)$, and the other one coinciding with $Odd\_atoms(C)$. Otherwise, it has just one stable model. (ii) if instead the cycle is odd, the program fragment has no stable models whenever the cycle is either unconstrained or actually unconstrained, and has just one stable model otherwise. In fact, if a handle assignment is effective then the corresponding program fragment is locally stratified, and thus (PP90) has a unique stable model that coincides with its well-founded model.

Then, it is easy to see that a non-effective handle assignment is never admissible for an odd cycle, and is admissible for an even cycle if and only if either $OUT_C^+ \subseteq Even\_atoms(C)$ or $OUT_C^+ \subseteq Odd\_atoms(C)$.

It may be also observed that a trivial handle assignment, which do not states requirements on the out-handles, is always admissible for even cycles, and it is admissible for odd cycles if it is effective.

The admissibility of a non-trivial effective handle assignment for cycle C can be checked syntactically, by means of the criterion that we state below. The advantage of this check is that it does not require to compute the well-founded model of $C + Aux(C) + ActivationAt_C(IN_C^A)$, but it just looks at the rules of C. Although the syntactic formulation may seem somewhat complex, it simply states in which cases an atom in OUT_C^+, which is required to be concluded true w.r.t. the given handle assignment (or, conversely, an atom in OUT_C^- which is required to be concluded false), is actually allowed to take the specified truth value without raising inconsistencies. Notice that OUT_C^+ and OUT_C^- must be mutually coherent, in the sense that truth of an atom in OUT_C^+ cannot rely on truth of an atom in OUT_C^- (that instead is required to be concluded false), and vice versa.

Proposition 1

A non-trivial effective handle assignment $\langle IN_C^A, IN_C^N, OUT_C^+, OUT_C^- \rangle$ to cycle C is admissible if and only if for every $\lambda_i \in OUT_C^+$ the following condition (1) holds, and for every $\lambda_k \in OUT_C^-$ the following condition (2) holds.

1. Condition 1.
   (a) Either there exists and OR handle h_o for λ_i, $h_o \in IN_C^A$ or
   (b) for every AND handle h_a for λ_i, $h_a \in IN_C^N$ and
   $\lambda_{i+1} \notin OUT_C^+$, and
   condition (2) holds for λ_{i+1}.
2. Condition 2.
   (a) For every OR handle h_o for λ, $h_o \in IN_C^N$, and
   (b) either there exists and AND handle h_a for λ such that $h_a \in IN_C^A$, or
   $\lambda_{k+1} \notin OUT_C^-$, and condition (1) holds for λ_{k+1}.

Proof

Let us first notice that the set of rules with head λ_i in $C + Aux(C) + ActivationAt_C(IN_C^A)$

consists of rule $\lambda_i \leftarrow$ *not* λ_{i+1}, Δ_i in C, and possibly, of one or more rules in $Aux(C)$. In fact, by the definition of canonical program, atoms in IN_C^A do not occur in C, and thus λ_i cannot belong to $ActivationAt_C(IN_C^A))$.

Consider an atom $\lambda_i \in OUT_C^+$, that we want to be concluded true in the partial stable model of C, which corresponds to the given handle assignment. For λ_i to be concluded true, there must be a rule whose conditions are guaranteed to be true w.r.t the handle assignment.

One possibility, formalized in Condition 1.(a), is that there exists an OR handle h_o for λ_i, $h_o \in IN_C^A$. That is, there is an auxiliary rule with head λ_i, and condition true w.r.t. the handle assignment.

Otherwise, as formalized in Condition 1.(b) we have to consider the rule of cycle C:

$$\lambda_i \leftarrow \textit{not } \lambda_{i+1}, \Delta_i$$

and check that all the conditions are guaranteed to be true by the handle assignment. First of all it must be $(\Delta_k : AND : \lambda_k) \in IN_C^N$ i.e., in the given handle assignment the AND handle referring to λ_i must be supposed to be not active, because an active AND handle makes the head of the rule false. Second, *not* λ_{i+1} must be true: this on the one hand requires $\lambda_{i+1} \notin OUT_C^+$, that would be a contradiction; on the other hand, requires λ_{i+1} to be concluded false. To this aim, condition (2), discussed below, must hold for λ_{i+1}.

Consider now an atom $\lambda_k \in OC_C^-$, that we want to be false the partial stable model of C, which corresponds to the given assignment: there must *not* be a rule for λ_k whose conditions all true w.r.t. the given assignment.

First, as formalized in Condition 2.(a), we must have any OR handle h_o for λ_k in IN_C^N. Otherwise, λ_k would be necessarily concluded true, being the head of an auxiliary rule with a true body.

Second, as formalized in Condition 2.(b), we also have to consider the rule of cycle C

$$\lambda_k \leftarrow \textit{not } \lambda_{k+1}, \Delta_k$$

and check that one of its two conditions is false w.r.t. the handle assignment. A first case is that $(\Delta_k : AND : \lambda_k) \in IN_C^A$, which means that the AND handle referring to λ_k is supposed to be active, i.e., false. Otherwise, *not* λ_{k+1} must be false, i.e., λ_{k+1} must be true. To this aim, provided that $\lambda_{k+1} \notin OUT_C^-$ (that would be a contradiction), condition (1) must hold for λ_{k+1}. □

The fact that Conditions 1 and 2 refer to each other is not surprising, since they are to be applied on cycles. Consider for instance the following cycle:

$e \leftarrow$ *not* f
$f \leftarrow$ *not* g
$g \leftarrow$ *not* e
$g \leftarrow h$

The handle assignment $\langle\{(h : OR)\}, \emptyset, \{g\}, \emptyset\rangle$ is admissible, since, according to Condition 1.(a), there exists an auxiliary rule with head g and body in IN_C^A. Also $\langle\{(h : OR)\}, \emptyset, \{g, e\}, \emptyset\rangle$ is admissible, because: g is as above; there is no OR handle for e, thus

Condition 1.(a) cannot be applied, but, considering rule $e \leftarrow$ *not* f (Condition 1.(b)), it is easy to see that Condition 2 holds of f, since there is no OR handle for f, and we have just shown that Condition 1 holds of g. Then, $\langle\{(h : OR)\}, \emptyset, \{g\}, \{e\}\rangle$ is not admissible, because Condition 2 does not hold for e.

It is important to notice that it is possible to determine admissible handle assignments from a partially specified one. An obvious way of doing that is guessing the missing sets, and checking whether the resulting handle assignment is admissible. It is however possible to do it much easily by exploiting the definitions.

For given IN_C's, it is easy to find the maximal values for OUT_C^+ and OUT_C^- that form an admissible handle assignment. If IN_C^A is empty, then they correspond to the stable models (if any) of the cycle taken by itself (without the auxiliary rules, since an empty IN_C^A empty means that no OR handle is active). If IN_C^A is not empty, by asserting the atoms in $ActivationAt_C(IN_C^A)$ as facts one computes the (unique) stable model of the extended cycle, and thus the maximal values for OUT_C^+ and OUT_C^-. These maximal values are determined by assuming all handles not belonging to the IN_C's to be not active.

Vice versa, given OUT_C^+ and OUT_C^-, and unknown or partially defined IN_C's, the conditions stated in Proposition 1 can be used for determining the subsets of H_C (incoming handles) that form admissible handle assignments.

Consider for instance the extended cycle:

$e \leftarrow$ *not* f, *not* r
$f \leftarrow$ *not* g
$g \leftarrow$ *not* e
$g \leftarrow v$
$g \leftarrow h$
$e \leftarrow s$
$e \leftarrow$ *not* h

Assume we let $OUT_C^+ = \{g\}$ and $OUT_C^- = \emptyset$. Then, for forming an admissible handle assignment we have three possibilities.

First, by Condition 1.(a) of Proposition 1, we can exploit the auxiliary rule $g \leftarrow v$, i.e. the handle $(v : OR)$, and let $IN_C^{A1} = \{(v : OR)\}$, and $IN_C^{N1} = \emptyset$.

Second, again by Condition 1.(a) of Proposition 1, we can exploit the other auxiliary rule $g \leftarrow h$, i.e. the handle $(h : OR)$, and let $IN_C^{A2} = \{(h : OR)\}$. This implies to insert into IN_C^{N2} the contrary and opposite handles, since they both occur in C, i.e. let $IN_C^{N2} = \{(h : AND), (\textit{not } h : OR)\}$.

Third, we can exploit condition 1.(b), and consider rule with head g in the cycle, i.e. $g \leftarrow$ *not* e, and verify Condition 2 for e, that must be false. For satisfying Condition 2.(a), we have to consider both the OR handles for e, i.e. handle $(\textit{not } h : OR)$ and handle $(s : OR)$, that must be included in IN_C^{N2}, i.e., $IN_C^{N3} = \{(\textit{not } h : OR), (s : OR)\}$. For satisfying Condition 2.(b) we have to consider rule $e \leftarrow$ *not* f, *not* r. Since we want g true, this implies f false, which means that for getting e false as well, we have to add its AND handle $(\textit{not } r : AND)$ to IN_C^{N3}. I.e., finally we get $IN_C^{N3} = \{(\textit{not } h : OR), (s : OR), (\textit{not } r : AND)\}$. This leads to add the opposite and contrary handles which occur in C to IN_C^{A2}, thus letting: $IN_C^{A3} = \{(h : OR)\}$.

We may notice that $IN_C^{A2} = IN_C^{A3}$ but $IN_C^{N2} \subseteq IN_C^{N3}$. Both choices form an admissible handle assignment, although the first one is more restricted. It turns out in fact that, in the above cycle, for building handle assignments where $OUT_C^+ = \{g\}$ and $OUT_C^- = \emptyset$, the handle (*not* $r : AND$) is actually irrelevant. This explains why the definition of handle assignment does not enforce one to set *all* handles as active/not active. We can introduce the following definition:

Definition 22
An admissible handle assignment $\langle IN_C^A, IN_C^N, OUT_C^+, OUT_C^- \rangle$ is *minimal* if there is no other sets $IN_C^{A'} \subset IN_C^A$ and $IN_C^{N'} \subset IN_C^N$ such that $\langle IN_C^{A'}, IN_C^{N'}, OUT_C^+, OUT_C^- \rangle$ is still admissible.

As we have seen above, there can be alternative minimal sets of incoming active handles for the same out-handles. However, there may also be the case there is none. There is for instance no possibility for $OUT_C^+ = \{g, f\}$, i.e., no choice for the IN_C's can produce a partial stable model where both g and f are true.

7 Cycle graphs

Finally, we build a graph whose nodes are the cycles, and whose edges are the handles. A handle is considered to connect the cycle it comes from, to the cycle(s) where the handle appears.

Paths on this graph (that we call *handle paths*) represent indirect connections between cycles through the handles. If the handles composing a path are supposed to be active, then the truth value of the atoms composing the cycle this path starts from influence the truth value of atoms in the subsequent cycles on the path. If the ending cycle of the path is an odd cycle, then its consistency may be guaranteed by this kind of influence.

We will formally define under which conditions the consistency of the odd cycles is guaranteed. At the end, we will state that finding handle paths for the odd cycles, and checking these conditions, is equivalent to checking the program for consistency.

As mentioned above, it is possible to uniquely identify the set $\{C_1, \ldots, C_w\}$ of the cycles that occur in program Π. This set can be divided in the two disjoint subsets of the even cycles $\{EC_1, \ldots, EC_g\}$, and of the odd cycles $\{OC_1, \ldots, OC_h\}$.

Then, the program structure in terms of cycles, handles and handle paths can be described by means of a graph, where cycles are the vertices and handles are the edges. Below in fact we introduce the novel notion of a cycle graph.

Definition 23
The Cycle Graph CG_Π, is a directed graph defined as follows:

- **Vertices.** One vertex for each of cycles $C_1, \ldots, C_w$. Vertices corresponding to even cycles are labeled as EC_i's while those corresponding to odd cycles are labeled as OC_j's.
- **Edges.** An edge (C_j, C_i) marked with $(\Delta : K : \lambda)$ for each handle $(\Delta : K : \lambda) \in H_{C_i}$ of cycle C_i, that comes from C_j.

Each marked edge will be denoted by $(C_j, C_i|\Delta : K : \lambda)$, where however by abuse of notation either (C_j or C_i or λ) will be omitted whenever they are clear from the context, and we may write for short $(C_j, C_i|h)$, h standing for a handle that is either clear from the context or does not matter in that point.

An edge on the CG connects the cycle a handle comes from to the cycle to which the handle belongs. Edges on the CG make it clear that handles *connect* different cycles: a handle Δ being or not being assumed to be active, corresponds to the atom α which occurs in Δ to be required to take a certain truth value in the cycles the handle comes from, depending of the kind of the handle. Precisely, if α is required to be true, then it must be concluded true in at least one of the cycles it is involved in. If α is required to be false, it must be concluded false in all cycles it is involved in.

As formally stated in Theorem 2, the odd cycles need to have at least one active handle, since on their own they would be inconsistent. If such a handle comes from another odd cycle, then we can repeat the same reasoning. Therefore, any odd cycle, for being consistent, must be directly or indirectly connected to some even cycle, through a "chains" of handles. On the CG, for every odd cycle it is possible to check whether such a connection may exists.

First, one has to check that any odd cycle OC has at least one handle. Secondly, one has to check that there may exist an admissible handle assignment for the cycle C the handle comes from, so as to make that handle active. This will imply to determine the IN_C's accordingly. Consequently, we have to iterate the reasoning for the handles of C. This means, we have to identify a subgraph of the CG where all the odd cycles are supported, while taking into account what follows. If a handle is active, its opposite and contrary handles are not, while two sibling handles are simultaneously active/not active. Each handle must be coherently considered active/non-active wherever it occurs on the edges of the CG.

Let us make this kind of reasoning formal.

Definition 24
Given program Φ, let a ***CG support set*** be a couple $S = \langle ACT^+, ACT^- \rangle$ of subsets of the handles marking the edges of CG_Φ, represented in the form $(\Delta : K)$ ($K = AND/OR$), where handles in ACT^+ are supposed to be active, and handles in ACT^- are supposed to be not active, and we have:
(i) $ACT^+ \cap ACT^- = \emptyset$.
(ii) neither ACT^+ nor ACT^- contain a couple of either opposite or contrary handles.
(iii) if two opposite handles h and h^- both occur on the CG, then ACT^+ contains handle h if and only if ACT^- contains its opposite handle h^-.
(iv) if two contrary handles h and h^n both occur on the CG, then ACT^+ contains handle h if and only if ACT^- contains its contrary handle h^n.
(v) if two sibling handles h and h^s both occur on the CG, then either $h, h^s \in ACT^+$ and $h, h^s \notin ACT^-$, or vice versa $h, h^s \in ACT^-$ and $h, h^s \notin ACT^+$

Given S, we will indicate its two components with $ACT^+(S)$ and $ACT^-(S)$. By abuse of notation, for the sake of readability we introduce some simplifying assumptions.

- Given handle $h = (\Delta : K : \lambda)$, by $ACT^+(S) \cup \{h\}$ (resp. $ACT^-(S) \cup \{h\}$) we mean $ACT^+(S) \cup \{(\Delta : K)\}$ (resp. $ACT^+(S) \cup \{(\Delta : K)\}$).

- Given handle $h \in ACT^+(S)$ (resp. $h \in ACT^-(S)$) of the form $(\Delta : K)$, by $IN_C^A \cup \{h\}$ (resp. $IN_C^N \cup \{h\}$) we mean: to identify the set $H = \{(\Delta : K : \lambda) \in H_C\}$ and perform $IN_C^A \cup H$ (resp. $IN_C^N \cup H$).
- By $H_C \cap ACT^+(S)$ (resp. $H_C \cap ACT^+(S)$) we mean $\{(\Delta : K : \lambda) \in H_C | (\Delta : K) \in ACT^+(S)\}$ (resp. $(\Delta : K) \in ACT^+(S)$).

A CG support set represents the handles that are supposed to be active/not active for making the odd cycles and the whole program consistent.

As discussed before, consistency is strongly conditioned by the odd cycles of the program. So, we have to restrict the attention on CG support sets including at least one active handle for each odd cycle, and then we have to check that the assumptions on the handles are mutually coherent, and are sufficient for ensuring consistency. An CG support set is *potentially adequate* if it provides at least one active incoming handle for each of the odd cycles.

Definition 25

An CG support set S is potentially adequate if for every odd cycle C in Π there exists a handle $h \in H_C$ such that $h \in ACT^+(S)$.

A CG support set S induces a set of handle assignments, one for each of the cycles $\{C_1, \ldots, C_w\}$ occurring in Π.

The induced assignments are obtained on the basis of the following observations:

- Each handle in $h \in ACT^+(S)$ is supposed to be active, and therefore it must be active for each cycle C_i such that $h \in H_{C_i}$.
- Each handle in $h \in ACT^-(S)$ is supposed to be not active, and therefore it must be not active for each of cycle C_j such that $h \in H_{C_j}$.
- If an handle h in S requires, in order to be active/not active, an atom β to be false, then it must be concluded false *in all the extended cycles of the program* h comes from.
- If an handle h in S requires, in order to be active/not active, an atom β to be true, then it must be concluded true *in all the extended cycles of the program* h comes from. This point deserves some comment, since one usually assumes that it suffices to conclude β true *somewhere* in the program. Consider however that any rule $\beta \leftarrow Body$ that allows β to be concluded true in some cycle is an auxiliary rule to all the other cycles β is involved into. This is why β is concluded true *everywhere it occurs.* This is the mechanism for selecting partial stable models of the cycles that agree on shared atoms, in order to assemble stable models of the overall program.

Definition 26

Let $S = \langle ACT^+, ACT^- \rangle$ be a GG support set which is potentially adequate. For each cycle C_k occurring in Π, $k \leq w$, the (possibly empty) handle assignment induced by this set is determined as follows.

1. Let $IN_{C_k}^A$ be $H_{C_k} \cap ACT^+(S)$.
2. Let $IN_{C_k}^N$ be $H_{C_k} \cap ACT^-(S)$.

3. Let $OUT^+_{C_k}$ be the (possibly empty) set of all atoms $\beta \in Out\_handles(C_k)$ such that there is a handle $h \in ACT^+(S)$ either of the form $(\beta : OR)$ or (*not* $\beta : AND$).
4. Let $OUT^-_{C_k}$ be the (possibly empty) set of all atoms $\alpha \in Out\_handles(C_k)$ such that there is a handle $h \in ACT^-(S)$ either of the form $(\alpha : AND)$ or (*not* $\alpha : OR$).
5. Verify that $OUT^-_{C_k} \cap OUT^+_{C_k} = \emptyset$.

If this is the case for each C_k, then S actually induces a set of handle assignments, and is called *coherent.* Otherwise, S does not induce a set of handle assignments, and is called *incoherent.*

The above definition does not guarantee that the assignments induced by a coherent support set are admissible, that the same atom is not required to be both true and false in the assignments of different cycles, and that the incoming handles of a cycle being supposed to be active/not active corresponds to a suitable setting of the out-handles of the cycles they come from. I.e., consider for instance cycle C_i which has an incoming handle, e.g. $h = (\beta : OR : \lambda)$, in $IN^A_{C_i}$: h is supposed to be active, which in turn means that β must be concluded true elsewhere in the program; then, for all cycles C_j where β is involved into, we must have $\beta \in OUT^+_{C_j}$, in order to fulfill the requirement. Of course, we have to consider both IN^A_C and IN^C_N, and both the AND and the OR handles.

The following definition formalizes this more strict requirements.

Definition 27

A coherent CG support set S of handle paths is *adequate* (w.r.t. not adequate) if for the induced handle assignments the following conditions hold:

1. they are admissible;
2. for each two cycles C_i, C_j in Π, $OUT^+_{C_i} \cap OUT^-_{C_j} = \emptyset$.
3. For every C_i in Π, for every handle $h \in IN^A_{C_k}$ of the form either $(\beta : OR : \lambda)$ or (*not* $\beta : AND : \lambda$), and for every handle $h \in IN^N_{C_k}$ of the form either $(\beta : AND : \lambda)$ or (*not* $\beta : OR : \lambda$), for every other cycle C_j in Π, $i \neq j$, such that $\beta \in Out\_handles(C_j)$, we have $\beta \in OUT^+_{C_j}$.
4. For every C_i in Π, for every handle $h \in IN^A_{C_k}$ of the form either (*not* $\beta : OR : \lambda$) or $(\beta : AND : \lambda)$, and for every handle $h \in IN^N_{C_k}$ of the form either (*not* $\beta : AND : \lambda$) or $(\beta : OR : \lambda)$, for every other cycle C_j in Π, $i \neq j$, such that $\beta \in Out\_handles(C_j)$, we have $\beta \in OUT^-_{C_j}$.

As we will prove later on, this set is adequate whenever the program is consistent, since the support provided by the handles in S allows every cycle to have partial stable models, and ensures that these partial stable models agree on shared atoms.

The above definitions allow us to define a procedure for trying to *find* adequate support sets starting from the odd cycles, and following the dependencies on the CG.

Definition 28 (*Procedure PACG for finding adequate CG support sets for program* Π)

1. Let initially $S = \langle \emptyset; \emptyset \rangle$.
2. For each cycle C_k occurring in Π, $k \leq w$, let initially $HA_{C_k} = \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$
3. For each odd cycle OC in Π do:

(a) Choose $h \in H_{OC}$. If $H_{OC} = \emptyset$, than FAIL.

(b) For chosen h:

i do $ACT^+(S) := ACT^+(S) \cup \{h\}$;

ii if h^s occurs in the CG, do $ACT^+(S) := ACT^+(S) \cup \{h^s\}$;

iii if h^- occurs in the CG, do $ACT^-(S) := ACT^-(S) \cup \{h^-\}$;

iv if h^n occurs in the CG, do $ACT^-(S) := ACT^-(S) \cup \{h^n\}$.

v For each cycle C_k in Π such that $h \in H_{C_k}$:

A do $IN^A_{C_k} := IN^A_{C_k} \cup h$;

B if h^s occurs in H_{C_j} for some cycle C_j (where possibly $j = k$), do $IN^A_{C_j} := IN^A_{C_j} \cup \{h^s\}$;

C if h^- occurs in H_{C_j} for some cycle C_j (where possibly $j = k$), do $IN^N_{C_j} := IN^N_{C_j} \cup \{h^-\}$;

D if h^n occurs in H_{C_j} for some cycle C_j (where possibly $j = k$), do $IN^N_{C_j} := IN^N_{C_j} \cup \{h^n\}$.

vi If h is either of the form $(\beta : OR)$ or $(\textit{not}\ \beta : AND)$, for each cycle C_k in Π where $\beta \in Out\_handles(C_k)$, do: $OUT^+_{C_k} := OUT^+_{C_k} \cup \{\beta\}$;

vii If h is either of the form $(\textit{not}\ \beta : OR)$ or $(\beta : AND)$, for each cycle C_k in Π where $\beta \in Out\_handles(C_k)$, do: $OUT^-_{C_k} := OUT^-_{C_k} \cup \{\beta\}$

4. REPEAT

(a) Verify that $ACT^+(S) \cap ACT^-(S) = \emptyset$. If not, FAIL.

(b) Verify that neither ACT^+ nor ACT^- contain a couple of either opposite or contrary handles. If not, FAIL.

(c) For each cycle C_k in Π such that $OUT^+_{C_k} \neq \emptyset$ or $OUT^-_{C_k} \neq \emptyset$:

i Verify that $OUT^+_{C_k} \cap OUT^-_{C_k} = \emptyset$. If not, FAIL.

ii Update (if needed) $IN^A_{C_k}$ and $IN^N_{C_k}$ w.r.t. $OUT^+_{C_k}$ and $OUT^-_{C_h}$, and check that the resulting handle assignment is admissible. If not, then FAIL.

iii For each other cycle C_h in Π do: verify that $OUT^+_{C_k} \cap OUT^-_{C_h} = \emptyset$, and that $OUT^-_{C_k} \cap OUT^+_{C_h} = \emptyset$. If not, FAIL.

(d) For each cycle C_k in Π, for each $h \in IN^A_{C_k}$:

i do $ACT^+(S) := ACT^+(S) \cup \{h\}$;

ii if h^s occurs in the CG, do $ACT^+(S) := ACT^+(S) \cup \{h^s\}$;

iii if h^- occurs in the CG, do $ACT^-(S) := ACT^-(S) \cup \{h^-\}$;

iv if h^n occurs in the CG, do $ACT^-(S) := ACT^-(S) \cup \{h^n\}$.

v For each cycle C_h in Π such that $h \in H_{C_h}$:

A do $IN^A_{C_h} := IN^A_{C_h} \cup h$;

B if h^s occurs in H_{C_j} for some cycle C_j (where possibly $j = h$), do $IN^A_{C_j} := IN^A_{C_j} \cup \{h^s\}$;

C if h^- occurs in H_{C_j} for some cycle C_j (where possibly $j = h$), do $IN^N_{C_j} := IN^N_{C_j} \cup \{h^-\}$;

D if h^n occurs in H_{C_j} for some cycle C_j (where possibly $j = h$), do $IN^N_{C_j} := IN^N_{C_j} \cup \{h^n\}$.

vi If h is either of the form $(\beta : OR)$ or $(not\ \beta : AND)$, for each cycle C_k in Π where $\beta \in Out\_handles(C_k)$, do: $OUT^+_{C_k} := OUT^+_{C_k} \cup \{\beta\}$;

vii If h is either of the form $(not\ \beta : OR)$ or $(\beta : AND)$, for each cycle C_k in Π where $\beta \in Out\_handles(C_k)$, do: $OUT^-_{C_k} := OUT^-_{C_k} \cup \{\beta\}$

(e) For each cycle C_k in Π, for each $h \in IN^N_{C_k}$:

i do $ACT^-(S) := ACT^-(S) \cup \{h\}$;

ii if h^s occurs in the CG, do $ACT^-(S) := ACT^-(S) \cup \{h^s\}$;

iii if h^- occurs in the CG, do $ACT^+(S) := ACT^-(S) \cup \{h^-\}$;

iv if h^n occurs in the CG, do $ACT^+(S) := ACT^+(S) \cup \{h^n\}$.

v For each cycle C_h in Π such that $h \in H_{C_h}$:

A do $IN^N_{C_h} := IN^N_{C_h} \cup h$;

B if h^s occurs in H_{C_j} for some cycle C_j (where possibly $j = h$), do $IN^N_{C_j} := IN^N_{C_j} \cup \{h^s\}$;

C if h^- occurs in H_{C_j} for some cycle C_j (where possibly $j = h$), do $IN^A_{C_j} := IN^A_{C_j} \cup \{h^-\}$;

D if h^n occurs in H_{C_j} for some cycle C_j (where possibly $j = h$), do $IN^A_{C_j} := IN^A_{C_j} \cup \{h^n\}$.

vi If h is either of the form $(\beta : OR)$ or $(not\ \beta : AND)$, for each cycle C_k in Π where $\beta \in Out\_handles(C_k)$, do: $OUT^-_{C_k} := OUT^-_{C_k} \cup \{\beta\}$;

vii If h is either of the form $(not\ \beta : OR)$ or $(\beta : AND)$, for each cycle C_k in Π where $\beta \in Out\_handles(C_k)$, do: $OUT^+_{C_k} := OUT^+_{C_k} \cup \{\beta\}$

UNTIL no set is updated by the previous steps.

Proposition 2

Procedure PACG either fails, or returns an adequate CG support set.

Proof

Whenever it does not fail, PACG clearly produces a CG support set S. In fact: points (i-ii) of Definition 24 are verified by steps 4.(a-b) of PACG; points (iii-v) of Definition 24 are enforced after any update to S, namely by steps 3.b.(ii-iv), 4.d.(ii-iv) and 4.e.(ii-iv). The CG support set S produced by PACG is potentially adequate by construction, since in step 3.a a handle for each odd cycle is included. S is also adequate, since in fact: admissible handle assignments for all cycles in Π are incrementally built and verified in steps 4.c.(i-ii), thus fulfilling point 1. of Definition 27. Point 2 of Definition 27 is verified in step 4.c.(iii). Finally, points 3-4 of Definition 27 are enforced by steps 3.b.(vi-vii), 4.d.(vi-vii) and 4.e.(vi-vii), after each update to the IN_C's of any cycle. □

We close this Section by proposing a comprehensive example to illustrate all the concepts we have introduced above. Consider the following collection of cycles.

— OC_1
$p \leftarrow$ *not* s, *not* c
$s \leftarrow$ *not* t
$t \leftarrow$ *not* p
$s \leftarrow a$

— OC_2
$q \leftarrow$ *not* q
$q \leftarrow$ *not* e

— OC_3
$r \leftarrow$ *not* r, *not* e

— EC_1
$a \leftarrow$ *not* c
$c \leftarrow$ *not* a

— EC_2
$a \leftarrow$ *not* b
$b \leftarrow$ *not* a

— EC_3
$e \leftarrow$ *not* f
$f \leftarrow$ *not* e

— OC'_2
$q \leftarrow$ *not* q
$q \leftarrow$ *not* e
$q \leftarrow a$

Consider program $\pi_1 = OC_1 \cup EC_1$. Its cycle graph is reported in Figure 1. The odd cycle OC_1 admits the unique potentially active handle (*not* $c : AND : p$) Then, we let S_{π_1} be such that $ACT^+(S_{\pi_1}) = \{(\textit{not}\ c : AND)\}$ and $ACT^-(S_{\pi_1}) = \emptyset$. The induced set of handle assignments are as follows.

For OC_1: $IN^A_{OC_1} = \{(\textit{not}\ c : AND)\}$, $OUT^+_{OC_1} = OUT^-_{OC_1} = \emptyset$. This assignment is trivially admissible, since there is no requirement on the out-handles.

For EC_1: $OUT^+_{EC_1} = \{c\}$, $OUT^-_{EC_1} = \{\emptyset\}$. $IN_{EC_1} = \emptyset$, since EC_1 is unconstrained. It is easy to verify that this handle assignment is admissible, by letting $\lambda_1 = a$ and $\lambda_2 = c$, where of course for c to be true a must be false. This handle assignment corresponds to selecting the partial stable model $\{c\}$ for EC_1, while discarding the other partial stable model $\{a\}$.

Then, S_{π_1} as defined above is an adequate CG support set.

Consider program $\pi_2 = OC_1 \cup EC_1 \cup EC_2$. The situation here is complicated as EC_1 and EC_2 are not independent. In fact, rule $a \leftarrow$ *not* b of EC_2 is an auxiliary rule for EC_1, and, vice versa, $a \leftarrow$ *not* c of EC_1 is an auxiliary rule for EC_2. Then, here we have a cyclic connection between the even cycles. This is evident on the cycle graph of π_2, reported in Figure 2.

The odd cycle OC_1 has two handles, of which at least one must be active. Let us first assume that $(not\ c : AND : p)$ is active. According to the PACG procedure, we try to assemble a CG support set S, by letting at first $ACT^+(S_{\pi_2}) = \{(not\ c : AND)\}$ and $ACT^-(S_{\pi_2}) = \{(not\ c : OR)\}$. In fact, since *not* c is an incoming OR handle for a in EC_2, when assuming $(not\ c : AND)$ to be active, we also have to assume its opposite handle and its contrary handle to be not active.

Consequently, we let $IN^A_{OC_1} = \{(not\ c : AND)\}$ and $IN^N_{EC_2} = \{(not\ c : OR)\}$ Now, we have to put $OUT^-_{OC_1} = \{p\}$ and $OUT^+_{EC_1} = \{c\}$. To form an admissible handle assignment for EC_1, this implies to let $IN^N_{EC_1} = \{(not\ b : OR)\}$. Consequently, we have to update $ACT^-(S_{\pi_2})$ which becomes: $ACT^-(S_{\pi_2}) = \{(not\ c : AND), (not\ b : OR)\}$. This leads to put $OUT^+_{EC_1} = \{b\}$.

By iterating the reasoning nothing changes, and thus the couple of sets $ACT^+(S_{\pi_2}) = \{(not\ c : AND)\}$ and $ACT^-(S_{\pi_2}) = \{(not\ c : OR), (not\ b : OR)\}$ form, as it is easy to verify, an adequate CG support set.

Notice that this kind of reasoning does not imply either to find the stable models of the cycles, or to consider every edge of the CG. In fact, we have had no need to consider the second incoming handle of OC_1.

Let us now make the alternative assumption, i.e. assume that $(a : OR : s)$ is active for OC_1. This means at first $ACT^+(S_{\pi_2}) = \{(a : OR)\}$ and $ACT^-(S_{\pi_2}) = \emptyset$, since *not* a does not occur in handles of the CG. This implies $OUT^+_{EC_1} = \{a\}$. Then, there is no requirement on IN_{EC_2} for forming an admissible handle assignment, and then the procedure stops here.

Consider program $\pi_3 = OC_1 \cup EC_1 \cup EC_2 \cup OC_2 \cup OC_3$. The situation here is hopeless, since the only incoming handles to OC_2 and OC_3 are opposite handles, that cannot be both active. This is evident on the cycle graph of π_3, reported in Figure 3. For the other cycles, the situation is exactly as before.

Then, there is a subprogram which is ok, and a subprogram which gives problems. We can fix these problems for instance by replacing OC_2 with OC'_2, thus obtaining program π_4 (CG in Figure 4) where we can exploit handle $(a : OR)$ for both OC_1 and OC'_2. It is easy to verify that the CG support set S composed of $ACT^+(S_{\pi_4}) = \{(a : OR), (not\ e : AND)\}$ and $ACT^-(S_{\pi_4}) = \{(not\ e : OR)\}$ is adequate. The need to support OC'_2 rules out the possibility of supporting OC_1 by means of the handle $(not\ c : AND : p)$.

8 Main result

For programs in canonical form, we can state the main result of the paper, which gives us a necessary and sufficient syntactic condition for consistency.

Theorem 3

A program Π has stable models if and only if there exists and adequate CG support set S for Π.

Proof

$\Leftarrow$

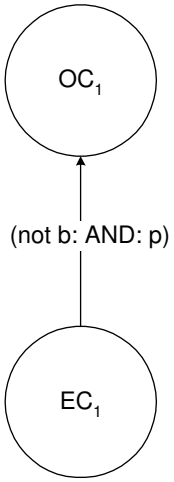

Figure 1. The Cycle Graph of π_1.

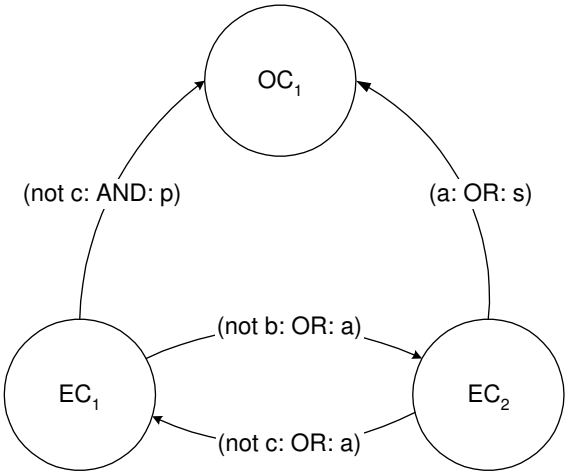

Figure 2. The Cycle Graph of π_2.

On the basis of S we can obtain the corresponding induced handle assignments, that will be admissible by the hypothesis that the S is adequate.

From S we can obtain a *global handle assignment* $HA = \langle T_{HA}; F_{HA} \rangle$ as follows.
$T_{HA} = \{\alpha |$
$(\alpha : OR) \in ACT^{+}(S) \vee (\mathit{not}\ \alpha : AND) \in ACT^{+}(S) \vee$
$(\mathit{not}\ \alpha : OR) \in ACT^{-}(S) \vee (\alpha : AND) \in ACT^{-}(S)\}$
$F_{HA} = \{\alpha |$
$(\alpha : AND) \in ACT^{+}(S) \vee (\mathit{not}\ \alpha : OR) \in ACT^{+}(S) \vee$
$(\alpha : OR) \in ACT^{-}(S) \vee (\mathit{not}\ \alpha : AND) \in ACT^{-}(S)\}$

Since the S is adequate, then by definition we have that: (i) by point 1 of Definition 27,

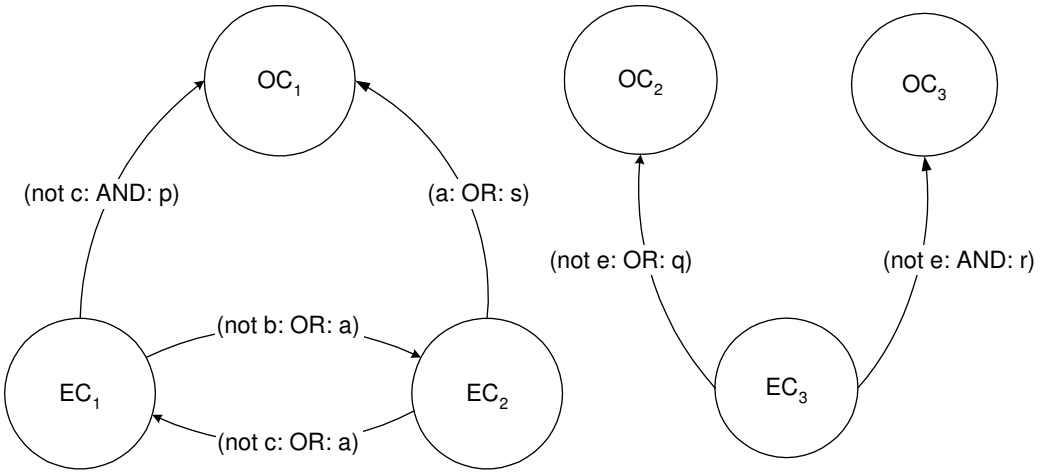

Figure 3. The Cycle Graph of π_3.

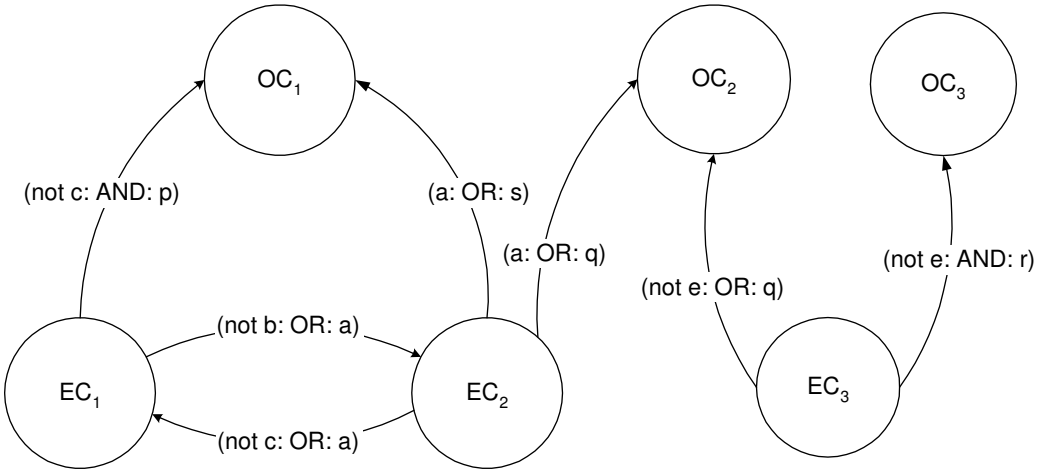

Figure 4. The Cycle Graph of π_4.

for each cycle C_i in Π, S induces an admissible handle assignment; (ii) HA is consistent, i.e. $T_{HA} \cap F_{HA} = \emptyset$, which is an immediate consequence of point 2 of Definition 27; (iii) by point 3 of Definition 27, $\forall \alpha \in T_{HA}$, α is concluded true in every cycle C_i it is involved into, since $\alpha \in OUT^+_{C_i}$, and the handle assignment induced by S to C_i is admissible; similarly, (iv) by point 4 of Definition 27, $\forall \alpha \in F_{HA}$, α is concluded false in every cycle C_j it is involved into, since $\alpha \in OUT^+_{C_j}$, and the handle assignment induced by S to C_j is admissible.

On the basis of HA, for each cycle C_i in Π we can build a correspondent independent program $C_i + Aux(C_i) + X_i$, where we let $X_i = Atoms(H_{C_i}) \cap T_{HA}$, and $X_i^- =$

$Atoms(H_{C_i}) \cap F_{HA}$. This independent program has a stable model S_i by construction, since the handle assignment induced by S to C_i is admissible (point (i) above). This stable model is (by Definition 15) a partial stable model for Π relative to C_i, with base $\langle X_i, X_i^- \rangle$. Taken one S_i for each C_i in Π, in the terminology of Definition 16 the S_i's constitute a compatible set of partial stable models because, according to Definition 16: (1) for any other cycle C_j, $X_i \cap X_j^- = \emptyset$, since $T_{HA} \cap F_{HA} = \emptyset$ by point (ii) above; (2) $\forall A \in X_i$, A is concluded true in some other cycle, by point (iii) above; (3) $\forall A \in X_i^-$, A is concluded false all the other cycles, by point (iv) above. Then, by Theorem 2, Π has stable models.

$\Rightarrow$

If Π has a stable model M, then by Theorem 2 we can decompose M into a compatible set of partial stable models, one partial stable model S_i, with base $\langle X_i, X_i^- \rangle$, for each C_i in Π. Since $X_i, X_i^- \subseteq Atoms(H_{C_i})$, they correspond to sets $IN_{C_i}^A$ and $IN_{C_i}^N$ of handles of C_i that are made active/not active by this base. By point (1) of Definition 16, for every two cycles C_i, C_j $X_i \cap X_i^- = \emptyset$, and then $IN_{C_i}^A \cap IN_{C_j}^N = \emptyset$. If we let S such that $ACT^+(S) = \bigcup_{i \leq w} IN_{C_i}^A$ and $ACT^-(S) = \bigcup_{i \leq w} IN_{C_i}^N$, we have $ACT^+(S) \cap ACT^-(S) = \emptyset$ and, if there are either opposite or contrary handles, they will not be in the same set. Then, S is a CG support set, and is potentially adequate by construction, because it has been built from the IN^A's and IN^N's of the cycles (point 1-2 of Definition 25), and because the S_i's agree on shared atoms, having been obtained by decomposing a stable model (points 3-5 of Definition 25). For the same reasons, S is also adequate.

□

Checking the condition stated in Theorem 3 does not imply finding the stable models of the program. However, in the proof of the *only-if* part, a way of determining the stable models is actually outlined, and is summarized below.

Corollary 3

Assume that the condition stated in Theorem 3 holds for program Π. Then, the stable models of Π can be determined as follows.

1. Given the handle assignments induced by S, build T_{HA}.
2. For each cycle C_i in Π, let $X_i = Atoms(H_{C_i}) \cap T_{HA}$, build the corresponding extended cycle $C_i + Aux(C_i) + X_i$, and find its partial stable models.
3. Assemble each stable model of Π as the union of one partial stable model for each cycle.

Let us reconsider the previous example. Let for short C^e be $C + Aux(C)$

For π_1, we have $T_{HA} = \{c\}$. The partial stable model of $EC_1^e \cup c$ is $\{c\}$; the partial stable model of $OC_1^e \cup c$ is $\{c, t\}$. Then, a stable model of the overall program is (as it is easy to verify) $\{c, t\}$.

For π_2, we have two possibilities. In the first one, $T_{HA} = \{c\}$. The partial stable model of $EC_1^e \cup c$ is $\{c\}$; the partial stable model of $EC_2^e \cup c$ is $\{c, b\}$; the partial stable model of $OC_1^e \cup c$ is $\{c, t\}$. Then, a stable model of the overall program is (as it is easy to verify) $\{c, b, t\}$.

In the second one, $T_{HA} = \{a\}$. The partial stable model of $EC_1^e \cup a$ is $\{a\}$; the partial stable model of $EC_2^e \cup a$ is $\{a\}$; the partial stable model of $OC_1^e \cup a$ is $\{a, s, t\}$. Then, a stable model of the overall program is (as it is easy to verify) $\{a, s, t\}$.

For π_4, we have $T_{HA} = \{a, e\}$. The partial stable model of $EC_1^e \cup a$ is $\{a\}$; the partial stable model of $EC_2^e \cup a$ is $\{a\}$; the partial stable model of $OC_1^e \cup a$ is $\{a, s, t\}$; the partial stable model of ${OC'}_2^e \cup a$ is $\{a, q\}$; the partial stable model of $EC_3^e \cup e$ is $\{e\}$; the partial stable model of $OC_3^e \cup e$ is $\{e\}$. Then, a stable model of the overall program is (as it is easy to verify) $\{a, s, t, q, e\}$.

9 Usefulness of the result

We believe that our results can be useful in different directions: (i) making consistency checking algorithms more efficient in the average case; (ii) defining useful classes of programs which are consistent by construction; (iii) introducing component-based software engineering principles and methodologies for answer set programming. This by defining, over the CG, higher level graphs where vertices are components, consisting of bunches of cycles, and edges are the same of the CG, connecting components instead of single cycles.

The three points are discussed at some length below.

9.1 Splitting consistency checking into stages

The approach and the result that we have presented here can lead to defining new algorithms for computing stable models. However, they can also be useful for improving existing algorithms.

We have identified and discussed in depth two aspects of consistency checking: (1) the odd cycles must be (either directly or indirectly) supported by the even cycles; (2) this support must be consistent, in the sense that no contrasting assumptions on the handles can be made.

Point (1) is related to the coarse" structure of the program, and can be easily checked on the CG, so as to rule out a lot of inconsistent programs, thus leaving only the potentially consistent" ones to be checked w.r.t. point (2). This is the aspect that might be potentially exploited by any approach to stable models computation.

Notice that a CG support set S determines a subgraph of the CG, which is composed of all the edges (and the corresponding end vertices) marked with the handles which occur in S.

Definition 29
Given the CG of program Π, and a CG support set S, an *adequate support subgraph* is a subgraph CG_S of the CG, composed of the edges marked by the handles belonging to $ACT^+(S)$ and $ACT^-(S)$, and of the vertices connected by these edges.

It is easy to see that, syntactically, CG_S is composed of a set of *handle paths,* that connect the odd cycles, through a chain of handles, to the even cycles (or to cyclic bunches of even cycles) that are able to support them. Each path may include more than one odd cycle, while each odd cycle must occur in at least one path.

Then, point 1 above may consist in checking whether a subgraph of the CG with this syntactic structure exists. Point 2, however performed, in essence must check whether the handles marking the subgraph constitute an adequate CG support set.

Staying within the approach of this paper, one may observe that the PACG procedure can easily be generalized for computing the stable models by performing the two steps in parallel. In fact, PACG it actually tries reconstruct the CG_S, starting from the odd cycles and going backwards through the CG edges to collect the handles that form the set S. At each step however, the procedure updates the handle assignments of the cycles and performs the necessary checks to be sure to be assembling an adequate set S. The extension would consist in computing the stable models of the extended cycles instead of just the handle assignments, and perform the computation on the whole CG.

9.2 Defining classes of programs that are consistent by construction

Based on the CG it is possible to define syntactic restrictions that, with a slight loss of expressivity, may ensure the existence of stable models. Suitable restrictions might be enforced on line by an automated tool, while the program is being written. This can be made easier by limiting the number of handles each cycle my have.

A first discussion in terms of cycles and handles about the most common situations where stable models exist or not is proposed in (CosPro03). The definition of classes of programs suitable for "interesting" applications is a topic of further research, but it can be useful to give some hints here.

In the literature, various sufficiency conditions have been defined (beyond stratification) for existence of stable models.

- Acyclic programs, by Apt and Bezem (AB91);
- Tight programs under certain conditions, by Erdem and Lifschitz (Erd99);
- Signed programs, by Turner (Tur94);
- Call-consistent programs, order consistent programs, and negative cycle free programs by Fages (Fag90), (Fag94).

We define below a new very simple class of programs that are guaranteed to have stable models, much broader than the above ones.

Definition 30
A program Π is called *tightly-even-bounded* if the following conditions hold: (i) every odd cycle has just one handle; (ii) this handle comes from an unconstrained even cycle; (ii) if there are two odd cycles whose handles come from the same even cycle, then two handles that originate in the same kind of node are of the same kind.

The above condition is clearly very easily and directly checked on the CG, and can be made clearly visible and understandable to a user, via a graphical interface. If you take any other existing graph representation, like e.g. the EDG (BCDP99) (Cos01), that computes stable models as *graph colorings*, the check is of course possible, but is less easy and less direct.

It is easy to see that:

Theorem 4

Every tightly even-bounded program P has stable models

Proof

Even cycles the handles come from are unconstrained, and conflicting handles are excluded by definition. Then, we can build an adequate CG support set by just assuming the incoming handles of the odd cycles to be active. □

Simple as it is, this is a class of non-call-consistent programs easy understandable by programmers, which is guaranteed to have stable models.

9.3 Generalizing the CG to components/agents

An important hot topic is, in our opinion, that of defining software engineering principles for Answer Set Programming.

Here we propose to define a program development methodology for Answer Set Programming by defining, over the CG, higher level graphs where vertices are *components,* and edges are the same of the CG, but connecting components instead of single cycles. We give below a first informal description of what kind of methodology we actually mean.

Let a *component* $\mathcal{C}$ be a bunch of cycles. It can be developed on its own, or it can be identified on the CG of a larger program. Similarly to a cycle however, $\mathcal{C}$ is not meant to be an independent program, but rather it has incoming handles.

As we have seen, partial stable models of cycles are characterized by handle assignments. Analogously, a component will be characterized by a *component interface*

$$IN_{\mathcal{C}}^{A}, IN_{\mathcal{C}}^{N}, OUT_{\mathcal{C}}^{+}, OUT_{\mathcal{C}}^{-}$$

that is meant to be a specification of which values the incoming handles may take, either in order to keep the component consistent, or in order to select some of its stable models. The out-handles provide the other components with a mean of establishing a connection with this one, i.e., they are true/false atoms that can make the incoming handles of other components active/not active as required.

Differently from cycles, in general components will not export *all* their active handles, but only those they want to make visible and available outside.

Based on the interface, it is possible to connect components, thus building a *Component Graph* $Comp\_G$. On this new graph $Comp\_CG$, one can either add new consistent components, or modify existing ones, and can check over the handle paths if there are problems for consistency, and how to fix them.

Referring to the previous example, in π_3 we have the component $OC_1 \cup EC_1 \cup EC_2$ which is consistent, and the component $OC_2 \cup OC_3 \cup EC_3$ which is instead inconsistent. Then, we have a $Comp\_CG$ with two unconnected vertices. We have shown how to fix the problem by adding a handle to OC_2, i.e., by suitably connecting the two components on the $Comp\_CG$.

In this framework, components may even be understood as independent agents, and making a handle active to a component may be understood as sending a message to the

component itself. Consider the following example, representing a fragment of the code of a *controller* component/agent:

$circuit\_ok \leftarrow not\ fault$
$fault \leftarrow not\ fault, not\ test\_ok$

where $test\_ok$ is an incoming handle, coming from a *tester* component/agent. As soon as the *tester* will achieve $test\_ok$, this incoming handle will become active, thus making the *controller* consistent, and able to conclude $circuit\_ok$.

A formal definition of the methodology we have outlined, and a detailed study of the applications, are the main future directions of this research.

References

Apt, K. and Bezem, M., *Acyclic Programs,* New Generation Computing 9 (3-4), 1991: 335–365

Balduccini M., Brignoli G., Lanzarone G.A., Magni F. and Provetti A., 2000. *Experiments in Answer Sets Planning,* Proc. of Mexican International Conference on Artificial Intelligence, 2000.

Brignoli, G., Costantini, S., D'Antona, O. and Provetti, A., 1999. *Characterizing and Computing Stable Models of Logic Programs: the Non–stratified Case,* Proc. of 1999 Conference on Information Technology, held in Bhubaneswar, India, December 1999.

Costantini, S., 1995. *Contributions to the Stable Model Semantics of Logic Programs with Negation,* Theoretical Computer Science 149, 1995 : 231-255.

S. Costantini, G. A. Lanzarone, G. Magliocco. *Asserting Lemmas in the Stable Model Semantics*. Logic Programming: Proc. of the 1996 Joint International Conference and Symposium (held in Bonn, Germany, September 1996). The MIT Press, USA, 1996.

Costantini, S., 2001. *Comparing different graph representations of logic programs under the Answer Set semantics,* Proc. AAAI Spring Symposium "Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning", Stanford, CA, March 26-28 2001.

Costantini, S. D'Antona, O. and Provetti, A. *On the Equivalence and Range of Applicability of Graph-based Representations of Logic Programs.* Information Processing Letters, Vol. 84, N. 2, December 2002.

Costantini, S., Intrigila, B. and Provetti, A. *Coherence of Updates in Answer Set Programming*. In: G. Brewka and P. Peppas (eds.), Proc. of the IJCAI-2003 Workshop on Nonmonotonic Reasoning, Action and Change, NRAC03 (Acapulco, Mexico, August 2003).

Costantini, S., and Provetti A., 2002. *Normal Forms for Answer Set Programming.* Submitted for publication.

J. Dix. *A Classification Theory of Semantics of Normal Logic Programs: I. Strong Properties*. Fundamenta Informaticae XXII(3): 227–255, 1995.

J. Dix. *A Classification Theory of Semantics of Normal Logic Programs: I. Weak Properties*. Fundamenta Informaticae XXII(3) : 257–288, 1995.

Dimopoulos Y., Nebel B. and Koehler J., 1997. *Encoding Planning Problems in Nonmonotonic Logic Programs,* Proc. of 1997 European Conference on Planning: 169–181.

Dimopoulos, Y. and Torres, A., 1996. *Graph theoretical structures in logic programs and default theories,* Theoretical Computer Science 170 (1996): 209–244.

Erdem, E. and Lifschitz, V.,1999. *Transformations of Logic Programs Related to Causality and Planning,* In: M. Gelfond, N. Leone and G. Pfeifer (eds.), Logic Programming and Nonmonotonic Reasoning: Proc. of 5th International Conference, LPNMR'99, LNAI 1730, Springer-Verlag, 1999: 107–116.

Lifschitz, V. and Turner, H., 1994. *Splitting a logic program,* In: Proceedings of the Eleventh International Conference on Logic Programming, The MIT Press, Cambridge, MA, 1994: 23-37.

Faber W., Leone N. and Pfeifer G., 1999. *Pushing Goal Derivation in DLP Computations,* In: M. Gelfond, N. Leone and G. Pfeifer (eds.), Logic Programming and Nonmonotonic Reasoning: Proc. of 5th International Conference, LPNMR'99, LNAI 1730, Springer-Verlag, 1999: 117–191.

Fages, F., 1990. Consistency of Clark's completion and existence of stable models. Technical Report 90-15, Ecole Normale Superieure, 1990.

Fages, F., 1994. *Consistency of Clark's Completion and Existence of Stable Models,* Methods of Logic in Computer Science, 1, 1994: 51-60.

Fitting, M. 1985. *A Kreepke-Kleene Semantics Logic Programs,* In: *Journal of Logic Programming*,2 (4): 295–312.

Gelfond, M. and Lifschitz, V., 1988. *The Stable Model Semantics for Logic Programming,* In: R. Kowalski and K. Bowen (eds.) Logic Programming: Proc. of 5th International Conference and Symposium: 1070–1080.

Gelfond, M. and Lifschitz, V., 1991. *Classical Negation in Logic Programming and Disjunctive Databases,* New Generation Computing 9, 1991: 365–385.

Konczak, K., Schaub, T., and Linke, T., 2003. *Graphs and colorings for answer set programming: Abridged report* In: M. De Vos and A. Provetti (eds.), Answer Set Programming: Advances in Theory and Implementation, ASP03. Volume 78 of The CEUR Workshop Proceedings Series. http://eur-ws.org/Vol-78/.

Konczak, K., Schaub, T., and Linke, T., 2003. *Graphs and colorings for answer set programming with preferences: Preliminary report* In: M. De Vos and A. Provetti (eds.), Answer Set Programming: Advances in Theory and Implementation, ASP03. Volume 78 of The CEUR Workshop Proceedings Series. http://eur-ws.org/Vol-78/.

Liberatore P., 1999. *Algorithms and Experiments on Finding Minimal Models.* Technical Report, University of Rome "La Sapienza".

Lifschitz V., 1999. *Answer Set Planning.* in: D. De Schreye (ed.) Proc. of the 1999 International Conference on Logic Programming (invited talk), The MIT Press: 23–37.

Linke, T., 2001. *Graph Theoretical Characterization and Computation of Answer Sets,* In: Proceedings of IJCAI 2001.

Linke, T., 2003. *Using nested logic programs for answer set programming* In: M. De Vos and A. Provetti (eds.), Answer Set Programming: Advances in Theory and Implementation, ASP03. Volume 78 of The CEUR Workshop Proceedings Series. http://eur-ws.org/Vol-78/.

Linke, T., 2003. *Suitable graphs for answer set programming* In: M. De Vos and A. Provetti (eds.), Answer Set Programming: Advances in Theory and Implementation, ASP03. Volume 78 of The CEUR Workshop Proceedings Series. http://eur-ws.org/Vol-78/.

Marek, W., and Truszczyński, M., 1999. *Stable Models and an Alternative Logic Programming Paradigm,* In: The Logic Programming Paradigm: a 25-Year Perspective, Springer-Verlag: 375–398.

Niemelä, I. 1999. *Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm,* Annals of Mathematics and Artificial Intelligence, 1999.

Przymusinska, H., and Przymusinski, T. C., *Semantic Issues in Deductive Databases and Logic Programs*. R.B. Banerji (ed.) *Formal Techniques in Artificial Intelligence, a Sourcebook*, Elsevier Sc. Publ. B.V. (North Holland), 1990.

CCALC: *http://www.cs.utexas.edu/users/mcain/cc*
DeReS: *http://www.cs.engr.uky.edu/ lpnmr/DeReS.html*
DLV: *http://www.dbai.tuwien.ac.at/proj/dlv/*
SMODELS: *http://www.tcs.hut.fi/Software/smodels/*

H. Turner. Signed logic programs. In *Proc. of the 1994 International Symposium on Logic Programming*, pages 61–75, 1994.

Van Gelder A., Ross K.A. and Schlipf J., 1990. *The Well-Founded Semantics for General Logic Programs,* Journal of the ACM Vol. 38 N. 3.