Modular Abstract Diagnosis

Marco Comini, Giorgio Levi and Giuliana Vitiello

Abstract

Modular abstract diagnosis shows that the abstract diagnosis method [5] does not need to be extended to perform the diagnosis in a modular way. We can verify and debug incomplete programs, once we have the specifications for the missing program components.

Keywords: Logic Programming, Declarative Diagnosis, Abstract Diagnosis.

Abstract diagnosis [6, 7, 4] is a generalization of declarative diagnosis [11, 9, 8], where we consider operational properties, i.e., *observables*, which are properties which can be extracted from a goal computation. In a modular abstract diagnosis we are concerned with programs composed of separate modules. Our theory of abstract diagnosis can be applied to modular diagnosis even if our concrete semantics *is not* OR-compositional, while usually an OR-compositional semantics was needed (see, for example, modular analysis in [2]).

Abstract diagnosis consists of comparing the actual and the intended behaviors of the program P w.r.t. the observable property α and determining the wrong program components, when the two behaviors are different. The formulation is parametric w.r.t. the property α considered in the specification and in the actual behavior. Assume $\mathcal{F}[\![\mathsf{P}]\!]$ be the concrete semantics of a program P (in our case the least fixed point of the s-semantics immediate consequences operator $\mathcal{T}_P[\![\mathsf{P}]\!]$) and α be an observable, i.e., an abstract domain related to the concrete domain of answer substitutions by a Galois insertion, whose abstraction function is α. The standard correctness and completeness properties can now be defined as follows.

Definition 1 *[5, 4] Let* P *be a program and* α *be an observable.*

1. P *is* partially correct *w.r.t.* $\mathcal{I}_\alpha$ *if* $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) \leq \mathcal{I}_\alpha$.
2. P *is* complete *w.r.t.* $\mathcal{I}_\alpha$ *if* $\mathcal{I}_\alpha \leq \alpha(\mathcal{F}[\![\mathsf{P}]\!])$.
3. P *is* totally correct *w.r.t.* $\mathcal{I}_\alpha$, *if* $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) = \mathcal{I}_\alpha$.

Note that we are comparing the abstraction of the concrete semantics and the specification. Abstract diagnosis determines the errors without requiring to determine the symptoms in advance. This is obtained by systematically deriving all the incorrect clauses

Marco Comini and **Giorgio Levi**. Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa , Italy. comini, levi@di.unipi.it, http://www.di.unipi.it/~comini, levi

Giuliana Vitiello. Dipartimento di Informatica ed Applicazioni, Università di Salerno, Baronissi (Salerno), Italy, giuvit@dia.unisa.it

and uncovered elements. Incorrect clauses and uncovered elements are defined in terms of the abstract immediate consequences operator,

$$\mathcal{P}_\alpha[\![P]\!](\mathcal{K}_\alpha) = \lambda p(\boldsymbol{x}).\, \tilde{\bigcup}\{D \mid\ c = p(\boldsymbol{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed clause of } P,\ \boldsymbol{z} = var(c),\ \text{for } i \in [1,n],\ \boldsymbol{y}_i = var(A_i),\ D = (\alpha(\{E\}) \,\tilde{\otimes}^{\boldsymbol{z}}_{\boldsymbol{y}_1}\, \mathcal{K}_\alpha(A_1) \,\tilde{\otimes}^{\boldsymbol{z}}_{\boldsymbol{y}_2} \cdots \tilde{\otimes}^{\boldsymbol{z}}_{\boldsymbol{y}_n}\, \mathcal{K}_\alpha(A_n))\,\tilde{\restriction}_{\boldsymbol{x}}\},$$

where P is a program in equational form[1], $\mathcal{K}_\alpha$ is an abstract denotation and $\tilde{\bigcup}$, $\tilde{\otimes}$ and $\tilde{\restriction}$ are the abstract join, meet and restriction operators (automatically derived for each observable α from the concrete operators) [5, 4].

Definition 2 *Let P be a program. If there exists an $\mathbb{A}$-element σ such that $\sigma \not\leq \mathcal{I}_\alpha$ and $\sigma \leq \mathcal{P}_\alpha[\![\{c\}]\!](\mathcal{I}_\alpha)$, then the clause $c \in P$ is* incorrect *on σ.*

Moreover, an $\mathbb{A}$-element σ is uncovered *if $\sigma \leq \mathcal{I}_\alpha$ and $\sigma \not\leq \mathcal{P}_\alpha[\![P]\!](\mathcal{I}_\alpha)$.*

The abstract immediate consequences operator can in general introduce an approximation. *Complete observables* are those observables such that $\alpha \circ \mathcal{P}[\![P]\!] = \mathcal{P}_\alpha[\![P]\!] \circ \alpha$, while for *approximate observables*[2] $\alpha \circ \mathcal{P}[\![P]\!] \leq \mathcal{P}_\alpha[\![P]\!] \circ \alpha$ holds. Complete observables (such as computed answers, correct answers, etc.) usually lead to infinite specifications (and therefore to non-effective definitions of incorrect clauses and uncovered elements). On the contrary, specifications are finite (and diagnosis is effective) in the case of approximate observables (such as *depth*(k) answers, types, groundness dependencies, etc.). However weaker result hold, because of approximation. Here are the basic results of abstract diagnosis.

Let α be an approximate or complete observable. Then

- If there are no incorrect clauses in P, then P is partially correct w.r.t. $\mathcal{I}_\alpha$.
- Let P be partially correct w.r.t. $\mathcal{I}_\alpha$. If there exists an uncovered $\mathbb{A}$-element, then P is not complete.
- Let α be complete observable and P be a complete program w.r.t. $\mathcal{I}_\alpha$. If there exists an incorrect clause in P, then P is not partially correct.
- Let α be a complete observable and assume $\mathcal{P}_\alpha[\![P]\!]$ has a unique fixpoint. If there are no uncovered $\mathbb{A}$-elements, then P is complete w.r.t. $\mathcal{I}_\alpha$.

As already said, in modular abstract diagnosis we are concerned with programs composed of separate modules. The idea is that of performing the diagnosis in a modular way, i.e., module by module. Modular analysis is usually based on an *OR*-compositional semantics. For example, the modular analysis framework in [2] is based on the *OR*-compositional version of the s-semantics [1]. Our concrete semantics *is not* *OR*-compositional and this is obviously true for all its abstractions. However, we can note that abstract diagnosis does not require to actually compute the abstract semantics, since it is simply based on one application of the abstract immediate consequence operator to the specification. The s-semantics immediate consequence operator is known to be *OR*-compositional

[1] Given any program clause $p(t) \leftarrow p_1(t_1), \ldots, p_n(t_n)$ its equational form is the formula $p(x) \leftarrow E, p_1(x_1), \ldots, p_n(x_n)$ where $E = \{x = t, x_1 = t_1, \ldots, x_n = t_n\}$ ($x, x_1, \ldots, x_n$ are new distinct variables).

[2] Approximate observables have noetherian domains.

(see for example [10]). The same result holds (by definition) for all the abstract immediate consequence operator corresponding to complete and approximate observables. The conclusion is that our theory of abstract diagnosis can directly be applied to modular diagnosis, as we will formally show in the following.

We assume a program P to be partitioned into *predicate-disjoint* modules [2], such that each predicate symbol is completely defined by a single module. Namely,

Definition 3 *A program partitioning* $P_1, \ldots, P_n$ *is* predicate-disjoint *if, for any* $i \neq j$, $\mathit{PredsDef}(P_i) \cap \mathit{PredsDef}(P_j) = \emptyset$, *where* $\mathit{PredsDef}(P) := \{p \mid p(\mathbf{t}) \leftarrow B \in P\}$.

Specifications $\mathcal{I}_\alpha^1, \ldots, \mathcal{I}_\alpha^n$ are associated to modules $P_1, \ldots, P_n$. Since the partition is predicate-disjoint, any $\mathcal{I}_\alpha^i$ is undefined for all the pure atoms with predicate *not* in $\mathit{PredsDef}(P_i)$ and then $\forall i \neq j.\ \mathit{dom}(\mathcal{I}_\alpha^i) \cap \mathit{dom}(\mathcal{I}_\alpha^j) = \emptyset$. It is worth noting that our definition of partitioning does not require a *hierarchical decomposition*, since mutual recursion between modules is possible.

The overall specification is $\mathcal{I}_\alpha = \mathcal{I}_\alpha^1 \mathbin{\widetilde{\sqcup}} \ldots \mathbin{\widetilde{\sqcup}} \mathcal{I}_\alpha^n$. A module P_i does not necessarily need to use all the other modules. Hence we introduce the operator *use* which gives those specifications which are *relevant* to a module P_i, i.e., $\mathit{use}(P_i) = \{\mathcal{I}_\alpha^j \mid \textit{module } P_i \textit{ uses (i.e., clause bodies contain) predicates which are defined by } P_j\}$[3]. P_i is a *basic* module if it does not use other modules, i.e., if $\mathit{use}(P_i) = \emptyset$.

The decomposition into modules allows us to define incorrect clauses and uncovered $\mathbb{A}$-elements in a (more efficient) modular way.

Definition 4 *Let* $P_1, \ldots, P_n$ *be a program partitioning and* c *be a clause in* P_i, *for some* $1 \leq i \leq n$. *If there exists an* $\mathbb{A}$*-element* σ *such that* $\sigma \not\leq \mathcal{I}_\alpha^i$ *and* $\sigma \leq \mathcal{T}_\alpha[\![\{c\}]\!](\mathcal{I}_\alpha^i \mathbin{\widetilde{\sqcup}} \widetilde{\bigsqcup} \mathit{use}(P_i))$, *then the clause* $c \in P$ *is* m-incorrect *on* σ.

c is m-incorrect on σ if it derives, from the (relevant part of the) intended semantics, an $\mathbb{A}$-element which is not in the module's intended semantics.

Definition 5 *Let* $P_1, \ldots, P_n$ *be a program partitioning and let* $\sigma = p(\mathbf{x}) \mapsto D$ *be an* $\mathbb{A}$*-element such that* $p \in \mathit{PredsDef}(P_i)$, *for some* $1 \leq i \leq n$. σ *is* m-uncovered *if* $\sigma \leq \mathcal{I}_\alpha^i$ *and* $\sigma \not\leq \mathcal{T}_\alpha[\![P_i]\!](\mathcal{I}_\alpha^i \mathbin{\widetilde{\sqcup}} \widetilde{\bigsqcup} \mathit{use}(P_i))$.

An $\mathbb{A}$-element in the intended semantics of a module is m-uncovered if there are no clauses in the module deriving it from the (relevant part of the) intended semantics.

It is worth noting that, in the above definitions of m-incorrect clauses and m-uncovered $\mathbb{A}$-elements, we compare two denotations which give a meaning only to those predicates which are defined inside the module and that we are only concerned with the specifications used by the module.

In Figure 1 we show the meta-interpreter implementation of the algorithms to determine m-incorrect clauses and m-uncovered $\mathbb{A}$-elements. They are parametric w.r.t. the observable (the parameter `Obs` in the two main procedures), whose operations have to be specified in a suitable module. Apart from being generic, the meta-interpreter is very similar to those proposed for declarative diagnosis (using the concept of oracle simulation). The main difference is that our does not need to start from symptoms. In fact, our oracle simulation just needs to be applied to finitely many pure atomic goals (generated by the call to `userdefined/2`). If the oracle returns finitely many answers to each query

[3]Note that $\widetilde{\bigsqcup} \mathit{use}(P_i) \leq \mathcal{I}_\alpha$ and that $\mathit{dom}(\widetilde{\bigsqcup} \mathit{use}(P_i)) \cap \mathit{dom}(\mathcal{I}_\alpha^i) = \emptyset$. Moreover, due to $\mathcal{T}_\alpha[\![P_i]\!]$ definition, $\mathcal{T}_\alpha[\![P_i]\!](\mathcal{I}_\alpha^i \mathbin{\widetilde{\sqcup}} \widetilde{\bigsqcup} \mathit{use}(P_i)) = \mathcal{T}_\alpha[\![P_i]\!](\mathcal{I}_\alpha)$.

```
incorrectClause(Obs, Mod, RealElem, Clause) :-
    userdefined(Mod, Atom),
    observableClause(Obs, Atom, ClauseE, Body, Clause),
    meetTheAnswers(Obs, Body, ClauseE, RealBodyEs),
    domainProject(Obs, Atom, RealBodyEs, RealEs),
    domainSingleton(Obs, RealE, RealEs),
    not(
        (observableAnswers(Obs, Atom, IntendedEs),
        domainSingleton(Obs, RealE, IntendedEs))    ),
    showAelement(Obs, Atom, RealE, RealElem).

uncoveredE(Obs, Mod, IntendedElem) :-
    userdefined(Mod, Atom),
    observableAnswers(Obs, Atom, IntendedEs),
    domainSingleton(Obs, IntendedE, IntendedEs),
    not( (
        findall( (R, ClauseE, Body),
            (observableClause(Obs, Atom, ClauseE, Body, _),R=Atom), Bodies),
        joinTheAnswers(Obs,  Atom, Bodies, RealEs),
        domainSingleton(Obs, IntendedE, RealEs)   ) ),
    showAelement(Obs, Atom, IntendedE, IntendedElem).
```

Figure 1: The main module of the modular diagnosis program

(i.e., if $\mathcal{I}_\alpha$ is finite), the meta-interpreter systematically derive all the incorrect clauses and uncovered $\mathbb{A}$-elements.

Let us briefly explain the base meta-interpreter code.

`incorrectClause/4)` As specified by our oracle simulation methodology we have to perform a top-down resolution step for every pure atomic goal (relevant to the currently diagnosed module `Mod`). Hence we generate an atom and choose a clause $\mathsf{H} \leftarrow \mathsf{D}, \mathsf{B}$ unifying with it

```
userdefined(Mod, Atom),
observableClause(Obs, Atom, ClauseE, Body, Clause),
```

then we (incrementally) get from the oracle a solution for (all the atoms in) the body of the clause and we build (step by step) the meet of D and the solutions. Then we project away all uninteresting variables from the abstract resulting equation.

```
meetTheAnswers(Obs, Body, ClauseE, RealBodyEs),
domainProject(Obs, Atom, RealBodyEs, RealEs),
```

Then we test if the oracle can provide an equivalent answer, i.e., if there exists any $\mathsf{x} \leq$ `RealEs` and $\mathsf{x} \not\leq$ `IntendedEs` (an incorrectness bug generated by the selected clause).

```
domainSingleton(Obs, RealE, RealEs),
```

```
not(
    (observableAnswers(Obs, Atom, IntendedEs),
    domainSingleton(Obs, RealE, IntendedEs))    ),
```

If such an x exists we simply use the pretty-printing utility `showAelement` to return it to the user.

`uncoveredE/3`) Dually to the previous case, we have to ask the oracle an abstract answer (`IntendedE`) for a pure atomic goal. Then we test if the oracle simulation can provide an equivalent answer, i.e., if there exists any $x \leq$ `IntendedEs` and $x \not\leq$ `RealEs` (an incompleteness bug). We perform another oracle simulation, but this time we have to use the entire program instead of a single clause. Then the (partial) results for all the clauses must be joined, before we can check whether $x \leq$ `RealEs`.

It is worth noting that meta-interpreter work with the PROLOG representation (rather than with the equational CLP representation) of the programs and (whenever possible) of the specifications. The complete sources of the meta-interpreter can be found in [3], where we also show the implementation of the modules for the complete observable *computed answers* and for the approximate observables *depth*(k) *answers* and $\mathcal{POS}$.

The following theorems show that the general results on complete and approximate observables of abstract diagnosis do apply to modular abstract diagnosis.

Theorem 6 *Let* $P = P_1 \cup \cdots \cup P_n$. *If there are no m-incorrect clauses in any module* P_i, *then* P *is partially correct w.r.t.* α.

Proof. By hypothesis, for any i, $\forall c \in P_i$. $\mathcal{P}_\alpha[\![\{c\}]\!](\mathcal{I}^i_\alpha \mathbin{\widetilde{\sqcup}} \widetilde{\bigsqcup} use(P_i)) \leq \mathcal{I}^i_\alpha$. Hence, for any i, $\mathcal{P}_\alpha[\![P_i]\!](\mathcal{I}_\alpha) \leq \mathcal{I}^i_\alpha$. Now, by definition of $\mathcal{P}_\alpha[\![\cdot]\!]$, $\mathcal{P}_\alpha[\![\cup P_i]\!](\mathcal{I}_\alpha) \leq \widetilde{\bigsqcup} \mathcal{I}^i_\alpha = \mathcal{I}_\alpha$. Hence $\mathcal{I}_\alpha$ is a pre-fixpoint of $\mathcal{P}_\alpha[\![P]\!]$ and then, since $\alpha(\mathcal{F}[\![P]\!]) \leq \mathcal{F}_\alpha[\![P]\!] = \mathit{lfp}\, \mathcal{P}_\alpha[\![P]\!]$, by Tarski's theorem $\alpha(\mathcal{F}[\![P]\!]) \leq \mathcal{I}_\alpha$. ■

Theorem 7 *Let* α *be a complete observable and* $P = P_1 \cup \cdots \cup P_n$ *be a complete program w.r.t.* $\mathcal{I}_\alpha$. *If, for some* i, *there exists an m-incorrect clause in* P_i, *then* P *is not partially correct.*

Proof. By completeness of P and Theorem 3.5 in [5], $\mathcal{I}_\alpha \leq \alpha(\mathcal{F}[\![P]\!]) = \mathcal{F}_\alpha[\![P]\!]$. Then, by monotonicity of $\mathcal{P}_\alpha[\![\cdot]\!]$, $\mathcal{P}_\alpha[\![P]\!](\mathcal{I}_\alpha) \leq \mathcal{P}_\alpha[\![P]\!](\mathcal{F}_\alpha[\![P]\!]) = \mathcal{F}_\alpha[\![P]\!]$. Thus, $\sigma \not\leq \mathcal{I}^i_\alpha$ and $\sigma \leq \mathcal{P}_\alpha[\![\{c\}]\!](\mathcal{I}_\alpha) = \mathcal{P}_\alpha[\![\{c\}]\!](\mathcal{I}^i_\alpha \mathbin{\widetilde{\sqcup}} \widetilde{\bigsqcup} use(P_i))$ implies $\sigma \not\leq \mathcal{I}^i_\alpha$ and $\sigma \leq \mathcal{F}_\alpha[\![P]\!] = \alpha(\mathcal{F}[\![P]\!])$, which means that P is not partially correct. ■

Theorem 8 *Let* $P = P_1 \cup \cdots \cup P_n$ *be partially correct w.r.t.* $\mathcal{I}_\alpha$. *If there exists an m-uncovered* $\mathbb{A}$*-element, then* P *is not complete.*

Proof. Let $\sigma = p(\mathbf{x}) \mapsto D$ with $p \in \mathit{PredsDef}(P_i)$, for some $1 \leq i \leq n$. By Theorem 3.5 in [5] and partial correctness of P, $\mathcal{F}_\alpha[\![P]\!] = \alpha(\mathcal{F}[\![P]\!]) \leq \mathcal{I}_\alpha$. By monotonicity of $\mathcal{P}_\alpha[\![P]\!]$,

$$\alpha(\mathcal{F}[\![P]\!]) = \mathcal{P}_\alpha[\![P]\!](\mathcal{F}_\alpha[\![P]\!]) \leq \mathcal{P}_\alpha[\![P]\!](\mathcal{I}_\alpha). \tag{1}$$

Now, $\sigma \leq \mathcal{I}^i_\alpha$ and $\sigma \not\leq \mathcal{P}_\alpha[\![P_i]\!](\mathcal{I}_\alpha) = \mathcal{P}_\alpha[\![P_i]\!](\mathcal{I}^i_\alpha \mathbin{\widetilde{\sqcup}} \widetilde{\bigsqcup} use(P_i))$ implies (since the other modules cannot define predicate p) $\sigma \leq \mathcal{I}^i_\alpha$ and $\sigma \not\leq \mathcal{P}_\alpha[\![P]\!](\mathcal{I}_\alpha)$. Thus $\sigma \leq \mathcal{I}^i_\alpha \leq \mathcal{I}_\alpha$ and, by (1), $\sigma \not\leq \alpha(\mathcal{F}[\![P]\!])$, which means that P is not complete. ■

Module "satisfiable"

```
sat(true).
sat(or(X,Y)) :- sat(X).
sat(neg(X)) :- inv(X).

%the clause sat(or(X,Y)) :- sat(Y). is missing
```

Module "invalid", this module *is not* supplied to the diagnoser.

```
inv(false).
inv(neg(X)) :- sat(X).
inv(or(X,Y)) :- inv(X),inv(Y).
```

Figure 2: The program of Example 10

Theorem 9 *Let α be a complete observable, $\mathsf{P} = \mathsf{P}_1 \cup \cdots \cup \mathsf{P}_n$ be a program and assume $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ has a unique fixpoint. If there are no m-uncovered $\mathbb{A}$-elements, then P is complete w.r.t. $\mathcal{I}_\alpha$.*

Proof. Absence of m-uncovered $\mathbb{A}$-elements implies $\forall \mathsf{i}.\, \mathcal{I}^{\mathsf{i}}_\alpha \leq \mathcal{P}_\alpha[\![\mathsf{P}_\mathsf{i}]\!](\mathcal{I}_\alpha) = \mathcal{P}_\alpha[\![\mathsf{P}_\mathsf{i}]\!](\mathcal{I}^{\mathsf{i}}_\alpha \,\widetilde{\sqcup}\, \widetilde{\bigsqcup} \mathit{use}(\mathsf{P}_\mathsf{i}))$. Hence, $\mathcal{I}_\alpha \leq \mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{I}_\alpha)$, i.e., $\mathcal{I}_\alpha$ is a post-fixpoint of $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$. Then, by Tarski's theorem, $\mathcal{I}_\alpha \leq \mathit{gfp}(\mathcal{P}_\alpha[\![\mathsf{P}]\!])$. Since, by Theorem 3.5 in [5], $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) = \mathcal{F}_\alpha[\![\mathsf{P}]\!] = \mathit{lfp}\, \mathcal{P}_\alpha[\![\mathsf{P}]\!]$ and, by hypothesis, $\mathit{gfp}(\mathcal{P}_\alpha[\![\mathsf{P}]\!]) = \mathit{lfp}(\mathcal{P}_\alpha[\![\mathsf{P}]\!])$, the program P is complete. ■

Example 10

The program in Figure 2 is a wrong version of a program verifying satisfiability of boolean formulas (built with *or* and *neg*) which has a missing clause. We consider the approximate observable $\mathit{depth}(2)$-answers (τ_2) where we replace all the subterms at depth greater than 2 by a fresh variable. The answers of the diagnosis meta-interpreter is

```
incorrectClause(depth(2), modSat, AE, C).

Any (other) answer for sat(_1710) wrt obs. depth(2) ?  sat(or(X,true)).

Any (other) answer for sat(_1710) wrt obs. depth(2) ?  sat(true).

Any (other) answer for sat(_2173) wrt obs. depth(2) ?  sat(neg(false)).

Any (other) answer for sat(_2173) wrt obs. depth(2) ?  sat(neg(neg(J))).

Any (other) answer for sat(_2173) wrt obs. depth(2) ?  sat(or(true,Y)).

Any (other) answer for sat(_2197) wrt obs. depth(2) ?  sat(neg(or(J,K))).

Any (other) answer for sat(_2197) wrt obs. depth(2) ?  sat(or(Y,neg(J))).

Any (other) answer for sat(_2197) wrt obs. depth(2) ?  sat(or(Y,or(J,K))).

Any (other) answer for sat(_2197) wrt obs. depth(2) ?  sat(or(neg(J),Y)).
```

```
Any (other) answer for sat(_2197) wrt obs. depth(2) ?  sat(or(or(J,K),Y)).

Any (other) answer for sat(_1830) wrt obs. depth(2) ?  no.

Any (other) answer for inv(_1780) wrt obs. depth(2) ?  inv(false).

Any (other) answer for inv(_1780) wrt obs. depth(2) ?  inv(neg(neg(J))).

Any (other) answer for inv(_1780) wrt obs. depth(2) ?  inv(neg(or(J,K))).

Any (other) answer for inv(_1780) wrt obs. depth(2) ?  inv(neg(true)).

Any (other) answer for inv(_1780) wrt obs. depth(2) ?  inv(or(false,neg(J))).

Any (other) answer for inv(_1780) wrt obs. depth(2) ?  inv(or(false,or(J,K))).

Any (other) answer for inv(_1780) wrt obs. depth(2) ?  inv(or(neg(J),false)).

Any (other) answer for inv(_1780) wrt obs. depth(2) ?  inv(or(or(J,K),false)).

Any (other) answer for inv(_1780) wrt obs. depth(2) ?  inv(or(false,false)).

Any (other) answer for inv(_1780) wrt obs. depth(2) ?  no.

no (more) solutions

uncoveredE(depth(2), modSat, AE).

AE = sat(or(_1620,or(_1621,_1622))) ;

AE = sat(or(_1620,neg(_1621))) ;

AE = sat(or(_1620,true)) ;

no (more) solutions
```

As resulting from the answers of the user[4], the specification of the module "sat" is

$$\begin{aligned}\mathcal{I}_{\tau_2} := \mathrm{sat}(x) \mapsto \big\{&\{x = \mathrm{true}\}, \{x = \mathrm{or}(y, \mathrm{true})\}, \{x = \mathrm{or}(\mathrm{true}, y)\},\\ &\{x = \mathrm{or}(\mathrm{neg}(\hat{x}), y)\}, \{x = \mathrm{or}(y, \mathrm{neg}(\hat{x}))\},\\ &\{x = \mathrm{neg}(\mathrm{false})\}, \{x = \mathrm{neg}(\mathrm{neg}(\hat{x}))\},\\ &\{x = \mathrm{neg}(\mathrm{or}(\hat{x}, \hat{y}))\}, \{x = \mathrm{or}(y, \mathrm{or}(\hat{x}, \hat{y}))\},\\ &\{x = \mathrm{or}(\mathrm{or}(\hat{x}, \hat{y}), y)\}\big\}\end{aligned}$$

[4]Note that *depth*(2)-variables, which represent any term in the concrete domain, are denoted as $\hat{x}, \hat{y}$, to distinguish them from *depth*(1)-variables, which represent only themselves in the concrete domain.

while the specification of the module "invalid" is

$$\mathcal{K}_{\tau_2} := inv(x) \mapsto \big\{\{x = false\},\{x = or(false, false)\},\{x = neg(true)\},\\ \{x = neg(neg(\hat{x}))\},\{x = neg(or(\hat{x},\hat{y}))\},\\ \{x = or(false, neg(\hat{x}))\},\{x = or(neg(\hat{x}), false)\},\\ \{x = or(false, or(\hat{x},\hat{y}))\},\{x = or(or(\hat{x},\hat{y}), false)\}\big\}$$

Hence we find out that there are no m-incorrect clauses and then (by Theorem 6) the program is partially correct. Furthermore, the following $\mathbb{A}$-element are m-uncovered.

1. $sat(x) \mapsto \{x = or(y, or(v, w))\}$,
2. $sat(x) \mapsto \{x = or(y, neg(z))\}$,
3. $sat(x) \mapsto \{x = or(y, true)\}$.

Then (by Theorem 8) the (entire) program is not complete.

The above results show that, if we split the program and the specification into modules, we can determine incorrect clauses and uncovered $\mathbb{A}$-elements by means of (more efficient) *modular* algorithms. We will now turn to the (more interesting) case where we consider a single module (all the other modules may be not implemented yet), and we want to debug it, under the assumption that all the other missing modules do satisfy their specifications (i.e., are totally correct). The diagnosis will still be based on Definitions 4 and 5. However we have to introduce a new definition of partial correctness and completeness for a single module.

The new definitions are of course given in terms of the concrete semantics of a module P_i, which can be determined from the clauses in P_i and from the concrete semantics of the (missing) modules used by P_i. Since these modules have not been implemented yet, we have only their abstract specifications. In order to reason about the correctness and completeness of P_i we need a concrete specification. Thus we assume that their concrete semantics is simply the concretization of the abstract specifications[5]. This is achieved by first defining the following concrete ($\mathcal{I}_\alpha$-augmented) immediate consequence operator:

$$\mathcal{P}^{\mathcal{I}_\alpha}[\![P_i]\!](\mathcal{K}) := \mathcal{P}[\![P_i]\!](\mathcal{K} \sqcup \bigsqcup \gamma(use(P_i))).$$

This operator is continuous. The concrete semantics of a program module P_i can be then defined as

$$\mathcal{F}^{\mathcal{I}_\alpha}[\![P_i]\!] := \mathcal{P}^{\mathcal{I}_\alpha}[\![P_i]\!]\uparrow\omega\,.$$

This leads to the new notions of partial correctness and completeness of a module w.r.t. the intended abstract semantics of the whole program.

Definition 11 *Let $P_1, \ldots, P_n$ be a program partitioning, let α be an observable, and let $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$ be an intended module semantics. A module P_i is*

1. m-partially correct *w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$ if $\alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![P_i]\!]) \leq \mathcal{I}^i_\alpha$.*
2. m-complete *w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$ if $\mathcal{I}^i_\alpha \leq \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![P_i]\!])$*

[5]Note that the concretization $\gamma(\mathcal{I}_\alpha)$ of the abstract specification $\mathcal{I}_\alpha$ is the maximal of all possible concrete specifications $\mathcal{I}$ which $\mathcal{I}_\alpha$ represents. Indeed, for any $\mathcal{I}$ such that $\alpha(\mathcal{I}) = \mathcal{I}_\alpha$, $\mathcal{I} \sqsubseteq \gamma(\mathcal{I}_\alpha)$.

3. P *is* m-totally correct *w.r.t.* $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$, *if* $\alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]) = \mathcal{I}^i_\alpha$

The proof of the theorems uses the following (continuous) abstract ($\mathcal{I}_\alpha$-augmented) immediate consequence operator and its fixpoint.

$$\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{K}_\alpha) := \mathcal{P}_\alpha[\![\mathsf{P}_i]\!](\mathcal{K}_\alpha \,\widetilde{\sqcup}\, \widetilde{\bigsqcup} \mathit{use}(\mathsf{P}_i)), \qquad \mathcal{F}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!] := \mathit{lfp}(\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]),$$

Theorem 12 *If there are no m-incorrect clauses in* P_i, *then* P_i *is m-partially correct w.r.t.* $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$.

Proof. By hypothesis, for any clause c in P_i, $\mathcal{P}_\alpha[\![\{\mathsf{c}\}]\!](\mathcal{I}^i_\alpha \,\widetilde{\sqcup}\, \widetilde{\bigsqcup} \mathit{use}(\mathsf{P}_i)) \leq \mathcal{I}^i_\alpha$. Hence $\mathcal{P}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}_\alpha) \leq \mathcal{I}^i_\alpha$. Now, by definition, $\mathcal{P}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}_\alpha) = \mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}^i_\alpha)$ and then $\mathcal{I}^i_\alpha$ is a prefixpoint of $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$. Then, since $\alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]) \leq \mathcal{F}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!] = \mathit{lfp}(\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]) \leq \mathcal{I}^i_\alpha$, P_i is m-partially correct w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$. ■

Theorem 13 *Let* α *be a complete observable and* P_i *be an m-complete module w.r.t.* $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$, *for some* $1 \leq i \leq n$. *If there exists an m-incorrect clause in* P_i, *then* P_i *is not m-partially correct.*

Proof. By m-completeness of P_i and since α is complete, $\mathcal{I}^i_\alpha \leq \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]) = \mathcal{F}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$. By monotonicity of $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$, $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}^i_\alpha) \leq \mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{F}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]) = \mathcal{F}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$. On the other hand, by definition, $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}^i_\alpha) = \mathcal{P}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}_\alpha)$. Thus, $\sigma \not\leq \mathcal{I}^i_\alpha$ and $\sigma \leq \mathcal{P}_\alpha[\![\{\mathsf{c}\}]\!](\mathcal{I}_\alpha)$ implies $\sigma \not\leq \mathcal{I}^i_\alpha$ and $\sigma \leq \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!])$, i.e., P_i is not m-partially correct. ■

Theorem 14 *Let* P_i *be m-partially correct w.r.t.* $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$, *for some* $1 \leq i \leq n$. *If there exists an m-uncovered* $\mathbb{A}$*-element in* P_i, *then* P_i *is not m-complete.*

Proof. The operator $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$ is correct. Indeed,

$(\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!] \circ \alpha)(\mathcal{K}) =$ [by $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$ definition]

$\mathcal{P}_\alpha[\![\mathsf{P}_i]\!](\alpha(\mathcal{K}) \,\widetilde{\sqcup}\, \widetilde{\bigsqcup} \mathit{use}(\mathsf{P}_i)) =$ [since α is additive and $\alpha\gamma = \mathit{Id}$]

$(\mathcal{P}_\alpha[\![\mathsf{P}_i]\!] \circ \alpha)(\mathcal{K} \sqcup \bigsqcup \gamma\, \mathit{use}(\mathsf{P}_i)) \geq$ [since $\mathcal{P}_\alpha[\![\mathsf{P}_i]\!]$ is correct]

$(\alpha \circ \mathcal{P}[\![\mathsf{P}_i]\!])(\mathcal{K} \sqcup \bigsqcup \gamma\, \mathit{use}(\mathsf{P}_i)) =$ [by $\mathcal{P}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]$ definition]

$(\alpha \circ \mathcal{P}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!])(\mathcal{K})$.

Hence,

$\alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]) =$

[since $\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]$ is a fixpoint of $\mathcal{P}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]$]

$\alpha(\mathcal{P}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!](\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!])) \leq$

[by the previous result]

$\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!])) \leq$

[by monotonicity of $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$ and m-partial correctness of P_i]

$\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}^i_\alpha) =$

[by definition]

$\mathcal{P}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}_\alpha)$.

Now, if σ is an m-uncovered $\mathbb{A}$-element (i.e., $\sigma \leq \mathcal{I}^i_\alpha$ and $\sigma \not\leq \mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}^i_\alpha)$), then $\sigma \leq \mathcal{I}^i_\alpha$ and $\sigma \not\leq \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!])$, i.e., P_i is not m-complete. ■

```
acc([b|Xs]) :- accept(Xs).

%the clause acc([]). is missing
```

Figure 3: The program of Example 16

Theorem 15 *Let α be a complete observable and assume that $\mathcal{P}_\alpha^{\mathcal{I}_\alpha}[\![P_i]\!]$ has a unique fixpoint (for some $1 \leq i \leq n$). If there are no m-uncovered $\mathbb{A}$-elements in P_i, then P_i is m-complete w.r.t. $\mathcal{I}_\alpha^1, \ldots, \mathcal{I}_\alpha^n$.*

Proof. Absence of m-uncovered $\mathbb{A}$-elements in P_i implies $\mathcal{I}_\alpha^i \leq \mathcal{P}_\alpha[\![P_i]\!](\mathcal{I}_\alpha)$. Since, by definition, $\mathcal{P}_\alpha^{\mathcal{I}_\alpha}[\![P_i]\!](\mathcal{I}_\alpha^i) = \mathcal{P}_\alpha[\![P_i]\!](\mathcal{I}_\alpha)$, $\mathcal{I}_\alpha^i \leq \mathcal{P}_\alpha^{\mathcal{I}_\alpha}[\![P_i]\!](\mathcal{I}_\alpha^i)$, i.e., $\mathcal{I}_\alpha^i$ is a post-fixpoint of $\mathcal{P}_\alpha^{\mathcal{I}_\alpha}[\![P_i]\!]$ and, by Tarski's theorem, $\mathcal{I}_\alpha^i \leq \mathit{gfp}(\mathcal{P}_\alpha^{\mathcal{I}_\alpha}[\![P_i]\!])$.

Now, by hypothesis, $\mathit{gfp}(\mathcal{P}_\alpha^{\mathcal{I}_\alpha}[\![P_i]\!]) = \mathit{lfp}(\mathcal{P}_\alpha^{\mathcal{I}_\alpha}[\![P_i]\!]) = \mathcal{F}_\alpha^{\mathcal{I}_\alpha}[\![P_i]\!] = \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![P_i]\!])$. Thus, $\mathcal{I}_\alpha^i \leq \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![P_i]\!])$, i.e., P_i is m-complete w.r.t. $\mathcal{I}_\alpha^1, \ldots, \mathcal{I}_\alpha^n$. ■

Example 16

The program P_{acc} in Figure 3 is a wrong version of a module of an automaton which recognizes the language $L = \{(ab)^n \mid n \geq 0\} \cup \{(ab)^n a \mid n \geq 0\}$. The answers of the meta-interpreter is

```
uncoveredE(depth(2), modAcc, AE).

Any (other) answer for acc(_3430) wrt obs. depth(2) ?  acc([]).

Any (other) answer for accept(_3680) wrt obs. depth(2) ?  accept([]).

Any (other) answer for accept(_3680) wrt obs. depth(2) ?  accept([a]).

Any (other) answer for accept(_3680) wrt obs. depth(2) ?  accept([a,B|J]).

Any (other) answer for accept(_3680) wrt obs. depth(2) ?  no.

AE = acc([]) ;

Any (other) answer for acc(_3430) wrt obs. depth(2) ?  acc([b]).

Any (other) answer for acc(_3430) wrt obs. depth(2) ?  acc([b,A|J]).

Any (other) answer for acc(_3430) wrt obs. depth(2) ?  no.

no (more) solutions

incorrectClause(depth(2), modAcc, AE, C).

no (more) solutions
```

As resulting from the answers of the user, the specification of the diagnosed module w.r.t. the *depth*(2)-answer observable (τ_2) is

$$\mathcal{I}_{\tau_2} := acc(X) \mapsto \big\{\{X = [\,]\}, \{X = [b]\}, \{X = [b, \hat{a}|\hat{X}]\}\big\},$$

while the specification of the other module is

$$\mathcal{K}_{\tau_2} := \mathtt{accept}(X) \mapsto \big\{\{X = []\}, \{X = [a]\}, \{X = [a, \hat{b}|\hat{x}]\}\big\}.$$

Hence we find out that P_{acc} is m-partially correct (without actually computing $\mathcal{F}^{\mathcal{I}_\alpha}[\![P_{acc}]\!]$) and that the $\mathbb{A}$-element $\mathtt{acc}(x) \mapsto \{x = []\}$ is m-uncovered. Then (by Theorem 14) the module is not m-complete (and the entire program is not complete).

Theorem 17 *Let $P = P_1 \cup \cdots \cup P_n$. If all the modules P_i are m-partially correct w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$, then P is partially correct w.r.t. $\mathcal{I}_\alpha$. Moreover, if P has a unique fixpoint and all the modules P_i are m-complete w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$, then P is complete w.r.t. $\mathcal{I}_\alpha$.*

Proof. We prove the two points separately.

partial correctness) Let $\mathcal{K} = \bigsqcup_i \mathcal{K}^i$. For any i,

$$\mathcal{P}^{\mathcal{I}_\alpha}[\![P_i]\!](\mathcal{K}^i) = \mathcal{P}[\![P_i]\!](\mathcal{K}^i \sqcup \bigsqcup \gamma\, use(P_i)) \geq \mathcal{P}[\![P_i]\!](\mathcal{K}^i)$$

Thus $\bigsqcup_i \mathcal{P}^{\mathcal{I}_\alpha}[\![P_i]\!](\mathcal{K}^i) \geq \mathcal{P}[\![P]\!](\mathcal{K})$ and the following facts hold.

$$\begin{array}{ll}
\alpha(\mathcal{F}[\![P]\!]) \leq & [\text{ by the previous result }] \\
\alpha(\bigsqcup_i \mathcal{F}^{\mathcal{I}_\alpha}[\![P_i]\!]) = & [\text{ by additivity }] \\
\widetilde{\bigsqcup_i} \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![P_i]\!]) \leq & [\text{ by m-partial correctness }] \\
\widetilde{\bigsqcup_i} \mathcal{I}^i_\alpha = \mathcal{I}_\alpha &
\end{array}$$

completeness) First of all note that, if the hole program P has a unique fixpoint, than also any module P_i must have a unique fixpoint and $\bigsqcup_i \mathcal{F}^{\mathcal{I}_\alpha}[\![P_i]\!] \sqsubseteq \mathcal{F}[\![P]\!]$. Thus the following facts hold.

$$\begin{array}{ll}
\mathcal{I}_\alpha = \widetilde{\bigsqcup_i} \mathcal{I}^i_\alpha \leq & [\text{ by m-partial completeness }] \\
\widetilde{\bigsqcup_i} \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![P_i]\!]) = & [\text{ by additivity }] \\
\alpha(\bigsqcup_i \mathcal{F}^{\mathcal{I}_\alpha}[\![P_i]\!]) \leq & [\text{ by the fixpoint uniqueness of } P\,] \\
\alpha(\mathcal{F}[\![P]\!]) &
\end{array}$$

■

References

[1] A. Bossi, M. Gabbrielli, G. Levi, and M. C. Meo. A Compositional Semantics for Logic Programs. *Theoretical Computer Science*, 122(1–2):3–47, 1994.

[2] M. Codish, S. K. Debray, and R. Giacobazzi. Compositional Analysis of Modular Logic Programs. In *Proc. Twentieth Annual ACM Symp. on Principles of Programming Languages*, pages 451–464. ACM Press, 1993.

[3] M. Comini. Sources of the abstract diagnosis meta-interpreters. Available at http://www.di.unipi.it/~comini/Projects/.

[4] M. Comini, G. Levi, M. C. Meo, and G. Vitiello. Proving properties of logic programs by abstract diagnosis. In M. Dams, editor, *Analysis and Verification of Multiple-Agent Languages, 5th LOMAPS Workshop*, volume 1192 of *Lecture Notes in Computer Science*, pages 22–50. Springer-Verlag, 1996.

[5] M. Comini, G. Levi, M. C. Meo, and G. Vitiello. Abstract diagnosis. *Journal of Logic Programming*, 1998. To appear.

[6] M. Comini, G. Levi, and G. Vitiello. Abstract debugging of logic programs. In L. Fribourg and F. Turini, editors, *Proc. Logic Program Synthesis and Transformation and Metaprogramming in Logic 1994*, volume 883 of *Lecture Notes in Computer Science*, pages 440–450. Springer-Verlag, 1994.

[7] M. Comini, G. Levi, and G. Vitiello. Efficient detection of incompleteness errors in the abstract debugging of logic programs. In M. Ducassé, editor, *Proc. 2nd International Workshop on Automated and Algorithmic Debugging, AADEBUG'95*, 1995.

[8] G. Ferrand. Error Diagnosis in Logic Programming, an Adaptation of E. Y. Shapiro's Method. *Journal of Logic Programming*, 4:177–198, 1987.

[9] J. W. Lloyd. Declarative error diagnosis. *New Generation Computing*, 5(2):133–154, 1987.

[10] P. Mancarella and D. Pedreschi. An Algebra of Logic Programs. In R. A. Kowalski and K. A. Bowen, editors, *Proc. Fifth Int'l Conf. on Logic Programming*, pages 1006–1023. The MIT Press, 1988.

[11] E. Y. Shapiro. Algorithmic program debugging. In *Proc. Ninth Annual ACM Symp. on Principles of Programming Languages*, pages 412–531. ACM Press, 1982.