ABSTRACT
DIAGNOSIS

Comini, Leni, Meo & Vitricola, Abstract Diagnosis, JLP 38, 1-3 (1989)
THE DIAGNOSIS APPROACH TO PROGRAM VERIFICATION

- declarative diagnosis (de-bugging) [of logic programs]
  [Shapiro 83, Penard 87, Lloyd 87]

\( P \) logic program
\( I \) specification \( a \) (intended declarative semantics of \( P \))
\( [P] \) (actual) declarative semantics of \( P \)

- a method to prove the correctness and completeness of \( P \) w.r.t. \( I \)
  - comparison between \( I \) and \( [P] \)
  - if they are different, locate the program bugs

- abstract diagnosis
  [Comini, Levi & Vitiello] META 84 - AFADE 84-85 - LISP 85

\( P \) logic program
\( d \) observable property
- any abstraction of sub-trees
\( I_d \) abstract specification (intended behavior w.r.t. \( d \))
\( [P]_d \) (actual) behavior of \( P \) w.r.t. \( d \)

- more concise than the declarative semantics
  (e.g. when \( d \) is computed environment)
- less concise than the declarative semantics
DEclarative Diagnosis

vs.

AbSTRACT Diagnosis

- Declarative Diagnosis
  - Declarative semantics vs. reference semantics
  - Two steps method
    - Symptoms detection (using testing techniques)
    - From symptoms to errors

- Abstract Diagnosis
  - The reference semantics can be any abstract semantics (including the declarative one)
  - No need to detect symptoms in advance
    - For some abstract semantics it may be non-effective
. The specification $I$ is the intended s-semantic of $P$

. The actual semantic $[[P]]$ is the semantics $[[P]] = F[[P]] = Tp\Psi$

- The two definition styles (bottom-up and top-down) of the s-semantic's and its properties are relevant to diagnosis
  - The theory of diagnosis is based on the bottom-up characterisation
  - The equivalent top-down characterisation allows us to implement the diagnoser by means of simple meta-interpreters
  - The modifying property allows us to consider goal-independent specifications
    - If $P$ is correct and complete w.r.t. the goal-independent specification $I$ then it behaves correctly for all the goals

. The above properties hold for the definitions of the s-semantic we will common in either diagnosis

. Our version of the s-semantic
  - self-model, with equations on the humorous condition as constraints
A COLLECTING S-SEMANTICS FOR CLP(+,-)

- program representation (for the sake of simplicity)
  pure atoms + constraints

**Example**

\[ P(s(x), y, [x|2]) :- q(x, s(w)), p(w, s(y), z) \]

is represented as

\[ P(x, y, z) :- x = s(x), z = [x|2], w = s(w), y = s(y), q(x, w), p(w, y, z) \]

- constraints = sets of equations

\[ E \]

set of equations

\[ \text{soln}(E) \]

set of solutions of \( E \)

(solution is a grounding substitution \( \theta \) such that all the equations in \( E \theta \) are identities)

\[ E_1, E_2 \]

sets of equations

\[ E_1 \leq E_2 \iff \text{soln}(E_1) \leq \text{soln}(E_2) \]

\[ \mathcal{C} \]

the set of all sets of equations (modulo equivalence)
The semantics maps each pair $G$ to an element of $\mathcal{P}(E)$ (the set of answer constraints).

- Partial order on $\mathcal{P}(E)$
  
  $E_1 \leq E_2$ iff for each element $E \in E_1$, there exists $E' \in E_2$, and such that $E \leq E'$

- $(\mathcal{P}(E), E)$ is a complete lattice
  
  - Top element: $\subseteq$
  - Bottom element: $\emptyset$
  - Least upper bound lub: set union
  - Greatest lower bound glb

  $$\text{glb}(\{E_1, E_2, \ldots\}, \{E_1', E_2', \ldots\}) = \{E_1 \cup E_1', E_2 \cup E_2', \ldots, E_2 \cup E_2', \ldots\}$$

  - $E_1 \cup E_2$ corresponds to unification and is transformation to solved form
  - Satisfiability check
  - "Normal form" for the equivalence class
- partial functions from most general atomic goals to elements of $\mathcal{P}(E)$

- partial order on interpretations

\[ I_1 \leq I_2 \text{ iff, for any most general atomic goal } A, \quad I_1(A) \subseteq I_2(A) \]

- \((\mathcal{Y}, \leq)\) is a complete lattice
The immediate consequence operator

\[ Tp(I)(p(X_1, \ldots, X_n)) = \text{lub} \left( \{ \varepsilon \} \right) \]

\[ p(X_1, \ldots, X_n) : \varepsilon_1, \ldots, \varepsilon_k \sqsubseteq q(X_1, X_2, \ldots, X_m) \]

Heredity clause of \( p \)

\[ I(q(X_1, X_2, \ldots, X_m)) = \varepsilon_1, \]

\[ \vdots \]

\[ I(q_m(X_1, X_2, \ldots, X_m)) = \varepsilon_m, \]

\[ E = \{ \varepsilon_1, \ldots, \varepsilon_k \} \]

\[ \varepsilon = \emptyset \{(E_1, \varepsilon_1, \ldots, \varepsilon_k) \mid x_1, \ldots, x_m \} \]

- The \( S \)-semantics in the interpretation

\[ F[i] = Tp(u) \]
**P**

\[ F(P) = T P F W \]

**S**

- **P is partially correct w.r.t. S**
  \[ F(P) \leq S \]

- **P is complete w.r.t. S**
  \[ S \leq F(P) \]

  - Termination issues are not addressed
  - The C-semantics is too abstract

**Observation**

\( p(x_1, \ldots, x_n) \)  
\( E \)

- An observation is a partial function \( \sigma \) which maps  
  \( p(x_1, \ldots, x_n) \) onto \( \{ E \} \)
  - Observations are assignments

- An observation \( \sigma \) is an **incorrectness symptom**  
  \[ \sigma \notin F(P) \text{ and } \sigma \notin S \]
  - an unknown constraint computed by \( P \) not in the specification

- An observation \( \sigma \) is an **incompleteness symptom**  
  \[ \sigma \in S \text{ and } \sigma \notin F(P) \]
  - an answer constraint in the specification which is not computed by \( P \)
Some symptoms are not actual "bugs"; they are just consequences of other "basic symptoms."

\[ q(x) := \neg p(x). \]

\[ S(p(x)) = \{ \{ x=a \} \} \]
\[ S(q(x)) = \{ \{ x=a \} \} \]

\[ F[p] (p(x)) = \emptyset \]
\[ F[p] (q(x)) = \emptyset \]

\[ \mathcal{O}_2(p(x)) = \{ \{ x=a \} \} \] and \[ \mathcal{O}_2(q(x)) = \{ \{ x=a \} \} \]
are both incompleteness symptoms, but \[ \mathcal{O}_2 \] is just a consequence of \[ \mathcal{O}_2 \]

- some bugs cannot be detected by looking at the symptoms
- on incompleteness may be hidden by an incompleteness and vice versa

\[ q(x) := \neg p(x). \]
\[ p(x);: = x=b. \]

\[ S(q(x)) = \{ \{ x=b \} \} \]
\[ F[p] (p(x)) = \{ \{ x=b \} \} \]
\[ F[p] (q(x)) = \{ \{ x=b \} \} \]

- then avoids only an incompleteness symptom
\[ \mathcal{O}_2(p(x)) = \{ \{ x=b \} \} \]

- if we fix this bug (by removing the second clause), we get an incompleteness symptom, since \[ F[p] (q(x)) = \emptyset \]

- symptom detection requires a fixpoint computation
• the problems with symptoms can be solved by basing the diagnosis on the detection of
  • incorrect clauses
  • uncovered observations
    • detects basic and hidden bugs
    • requires one application of $T_p$ rather than a fixpoint computation
    • symptoms can be ignored

**incorrect clause**

The clause $c \in P$ is incorrect on the observation $\omega$ if

$$\omega \not\in S \quad \text{and} \quad \omega \not\in T_p(S)$$

(the clause $c$ derives a wrong answer constraint from the specification)

**uncovered observation**

The observation $\omega$ is uncovered if

$$\omega \in S \quad \text{and} \quad \omega \not\in T_p(S)$$

(here are no clauses in $P$ which can derive $\omega$ from the specification)

• we will show that the diagnosis can be based on the detection of incorrect clauses and uncovered observations
Theorem 1

If there are no incorrect clauses, then the program is partially correct:

- If there are no incorrect clauses, $T_P(S) \leq S$
- $S$ is a pre-fixpoint of $T_P$
- $F[P]$ is the least pre-fixpoint of $T_P \Rightarrow F[P] \leq S$

(if the program is not partially correct, then there exists an incorrect clause)

Theorem 2

If the program is partially correct, then it is not always the case that there are no incorrect clauses

(on incompleteness may be hidden by an incompleteness)

Theorem 3

If the program is complete, then

- If there exists an incorrect clause on $S$, then $S$ is an incompleteness symptom

- Incorrect clauses are more meaningful than incompleteness symptoms

- Presence of incorrect clauses implies partial correctness

- An incorrect clause always corresponds to a bug, while incompleteness may not always be a symptom
DIAGNOSIS OF COMPLETE\textbf{NESS}

- cannot in general be based on the detection of uncorrected observations

\textbf{Theorem}

\begin{align*}
\text{Then exist a program } P \text{ and a specification } S, \text{ such that:} \\
\text{. } & \text{there are no uncorrected observations} \\
\text{. } & P \text{ is not complete w.r.t. } S \\
\end{align*}

\begin{align*}
P & \quad p(x) :- p(x) . \\
S(p(x)) & = \{ \text{false} \} \\
F[p](p(x)) & = \varnothing \\
T_P(S)(p(x)) & = \{ \text{true} \} \\
\end{align*}

- \text{ } S \text{ is a fixpoint of } T_P \text{ different from the least fixpoint } F[l[P]] \\
- \text{ } \text{clearly related to loops}

- \text{A theorem similar to Theorem 1 holds, if we assume } T_P \text{ to have a unique fixpoint}
Theorem 4

If $I$ has a unique fixpoint and there are no uncovered observations, then the program is complete.

(Theorem 4: If $I$ has a unique fixpoint and $p$ is not complete, then there exists an uncovered observation)

Theorem 5

If the program is complete, then it is not always the case that there are no uncovered observations.

(An incompleteness may be hidden by an incorrectness)

Theorem 6

If the program is partially correct, then

- if there exists an uncovered observation $o$, then
- $o$ is an incompleteness symptom

- uncovered observations are more meaningful than incompleteness symptoms

- absence of uncovered observations implies completeness (if $I$ has one fixpoint only)

- an uncovered observation always corresponds to a bug, while
- this is not the case for incompleteness symptoms
acceptable programs were introduced to study the termination of pure PROLOG programs (Apt-Pedrini, '83)

- they are exactly the left-terminating programs (all the L-derivations for ground goals are finite)

- acceptability is undecidable, but

  all sensible programs turn out to be acceptable
  * all the pure PROLOG programs in the Sterling & Shapiro book are acceptable
  * most wrong versions of sensible programs are acceptable (unless the things are just related to termination)

- the ground immediate consequence operator has one fixpoint only

- we have proved that the same result hold for the semantics Tp
AN EXAMPLE

\[
\text{ancestor}(x, y) :\neg \text{parent}(y, x), \text{ancestor}(x, z).
\]

\[
\text{ancestor}(x, y) :\neg \text{parent}(x, y).
\]

--- missing basic hypotheses ---

\[
S(\text{parent}(x, y)) = \{\{x = \text{tread}, y = \text{abraham}\}, \{x = \text{abraham}, y = \text{isaac}\}\}
\]

\[
S(\text{ancestor}(x, y)) = \{\{x = \text{tread}, y = \text{abraham}\}, \{x = \text{abraham}, y = \text{isaac}\},
\{x = \text{tread}, y = \text{isaac}\}\}
\]

\[
F[[P]](\text{parent}(x, y)) = \emptyset
\]

\[
F[[P]](\text{ancestor}(x, y)) = \emptyset
\]

\[
T_P(S)(\text{parent}(x, y)) = \emptyset
\]

\[
T_P(S)(\text{ancestor}(x, y)) = \{\{x = \text{abraham}, y = \text{tread}\},
\{x = \text{isaac}, y = \text{abraham}\},
\{x = \text{tread}, y = \text{isaac}\}\}
\]

- The second clause is incorrect on the observations

  \[
  O_1(\text{ancestor}(x, y)) = \{\{x = \text{isaac}, y = \text{abraham}\}\}
  \]

  \[
  O_2(\text{ancestor}(x, y)) = \{\{x = \text{abraham}, y = \text{tread}\}\}
  \]

- The uncovered observations are

  \[
  O_3(\text{parent}(x, y)) = \{\{x = \text{tread}, y = \text{abraham}\}\}
  \]

  \[
  O_4(\text{parent}(x, y)) = \{\{x = \text{abraham}, y = \text{isaac}\}\}
  \]

  \[
  O_5(\text{ancestor}(x, y)) = \{\{x = \text{tread}, y = \text{abraham}\}\}
  \]

- There are no uncovered symptoms, even if there is a wrong clause

  \[
  T_P(\text{ancestor}(x, y)) = \{\{x = \text{tread}, y = \text{isaac}\}\}
  \]

- If it is an incompleteness symptom

  \[
  O_6(\text{ancestor}(x, y)) = \{\{x = \text{abraham}, y = \text{isaac}\}\}
  \]
A MORE SYMMETRIC FORMULATION OF DIAGNOSIS

- proposed by (Perera 1883) for deductive diagnosis
- extends to abstract diagnosis

- the specification is a pair \( (S^+, S^-) \)
  \[
  S^+ \quad \text{intended s-semantics (\( \text{efp}(T_p^s) \))}
  
  S^- \quad \text{intended \( \text{gfp}(T_p^s) \)}
  \]

- a new definition of completeness

  \[ P \text{ is complete w.r.t. } (S^+, S^-) \text{ if } \]
  \[ S^- \leq \text{gfp}(T_p^s) \]

- the new definition of uncovered domination (\( \text{cover} \))

  \[ \text{the domination } \sigma \text{ is uncovered if } \]
  \[ \sigma \leq S^- \text{ and } \sigma \not\in T_p^s(S^-) \]

- the completeness theorem

  if there are no uncovered obstructions, then the program is complete

- no need for the unique fixpoint assumption

  reduces to the standard definition under the unique fixpoint assumption

- require a more complex specification
**Oracle-based Top-down Diagnosis**

**Follow-up diagnosis**

- comparison between $S$ and $Tp(S)$
- $S$ must be specified in advance

**Top-down diagnosis**

- $S$ can be implemented by an oracle (querying the user)

$$\mathcal{R}(p(x_1, \ldots, x_n)) = \{ E \mid E \text{ is an intended query constraint of } \neg p(x_1, \ldots, x_n) \}$$

- The diagnosis is expressed in terms of oracle simulation
  - one resolution step with program clauses
  - answers to the remaining goals from the oracle
The immediate consequence operator

\[ \delta_p \]

Resolution in

\[ \begin{cases} \mathcal{A} \\ \mathcal{A'} \end{cases} \]

Answers from the oracle

Composition of answer constraints

- The S-semantics is the interpretation

\[ F[p] = Tp \uparrow \mathcal{W} \]

\[ \delta_p = Tp (\cup \mathcal{A}) = Tp (S) \]
The clause $c$ is incorrect on the observation $s$ iff
\[ s \subseteq S_{12} \text{ and } s \notin \bar{A} \]

**Theorem**

The observation $s$ is uncovered iff
\[ s \subseteq A \text{ and } s \notin S_0 \]

- can be implemented as metainterpreters
  - one oracle only
  - no need to start from symptoms
    - we only need to start from (finitely many)
      most frequent atomic goals
  - more efficient interaction with the user is possible
    - oracles for conjunctive "instrumented" goals
- the diagnosis is not effective, unless the intended $\mathcal{S}$-semantics $\mathcal{S}$ is finite
  - the bottom-up diagnosis is impossible, if $\mathcal{S}$ is infinite
  - in the top-down diagnosis, the oracle may return infinitely many answers to some queries

- we need finite approximations
  - partial specifications [Comini, Len & Vitello, IJPS 85]
  - widening techniques
  - finite) algebraic domains
• the property we consider
  • in the specification $S_0$
  • in the actual semantics of $a$
  • an observable $d$

• any abstraction of $S_0$-tests

• abstract diagnosis is based on a semantic framework when observables (and the corresponding semantics) are related (and formally derived) using abstract interpretation theory [Comini, Levi 1994; Comini, Levi 1995]

• a taxonomy of observables
  • each class has suitable precision and confluence properties

• the kernel semantics collects $S_0$-tests
  • here we will take the (most abstract) $S$-semantics as collecting semantics
The observable $d$ is a (relax insertion between $P(E)$ and an abstract domain $(D, \leq)$.

The observable must have a correct abstract immediate consequence operator $T^d$, whose least fixpoint is the abstract semantics $Fd[P]$.

The theory of diagnosis is based on the fixpoint semantics.

The observable must be conducting, i.e., the abstract behavior for any path $\pi$ must be uniquely determined by the path $\pi$-independent denotation $d(F[\pi])$.

If this is not the case, we cannot specify the path-independent behavior only.

There exist two classes of observables for which the above properties hold:

- Precise observables, for which $d(F[\pi]) = F_d[\pi]$.
  - All the results of diagnosis w.r.t. computed answers hold.
  - Useful to reconstruct declarative diagnosis, when specifications can still be infinite.

- Approximate observables, for which $d(F[\pi]) \leq F_d[\pi]$.
  - Includes finite domains such as $\text{depth}(\pi)$, $\text{POS (groundmen)}$ and various models and types domains.
FROM THE $S$-SEMANTICS TO THE ABSTRACT SEMANTICS

- an abstract domain $(\mathcal{D}_d, \leq)$ complete lattice
- a Galois insertion $(\delta, \varepsilon)$ between $(\mathcal{B}(E), \leq)$ and $(\mathcal{D}_d, \leq)$ satisfying the axioms of approximate observables

- the abstract immediate consequence operator $T_p^d$

  For most general atom $p(X_1, \ldots, X_m)$

  $T_p^d(I)(p(X_1, \ldots, X_m)) = \\
  \text{lub}_d \{ \varepsilon \}$

\[
p(X_1, \ldots, X_m) : -e_1, \ldots, e_k \sqcap q_e(x_1^2, \ldots, x_k^2), \ldots, q_m(X_1^m, \ldots, X_k^m)
\]

- named clause of $p$

\[
I(q_e(x_1^2, \ldots, x_k^2)) = \varepsilon_1,
\]

\[
I(q_m(X_1^m, \ldots, X_k^m)) = \varepsilon_m,
\]

\[
E = \{ e_1, \ldots, e_k \}
\]

\[
E = \text{gll}_{\alpha}(\{ \varepsilon \}, \varepsilon_1, \ldots, \varepsilon_k | x_1^2, \ldots, x_k^2)
\]

- the abstract semantics $F_d[p] = T_p^d\uparrow \omega$ satisfies

\[
d(F[d][p]) \leq F_d[p]
\]

- can be also computed as standard semantics of the "abstract program" (Ciarrochi, Achray & Lelièvre, JLP 94)
\[ \text{Diagnosis W.R.T. Approximate Observables} \]

\[ F_{d}[\llbracket P \rrbracket] = T_{p}^{d} \downarrow \omega \]
\[ d(F[\llbracket P \rrbracket]) = d(T_{p}^{d} \downarrow \omega) \in F_{d}[\llbracket P \rrbracket] \]
\[ S_{a} \]

- \text{P is partially correct w.r.t. } S_{a}
  \[ d(F[\llbracket P \rrbracket]) \leq_{a} S_{a} \]

- \text{P is complete w.r.t. } S_{a}
  \[ S_{a} \leq_{a} d(F[\llbracket P \rrbracket]) \]

**Incorrect Clause**

The clause \( c \in P \) is incorrect on the (abstract) notation \( \omega \) if
\[ \omega \not\models S_{a} \text{ and } \omega \leq_{a} T_{\delta}^{d}(S_{a}) \]
(\( c \) denies a wrong abstract constraint from the specification)

**Uncovered Observation**

The (abstract) notation \( \omega \) is uncovered if
\[ \omega \leq_{a} S_{a} \text{ and } \omega \not\models_{a} T_{\delta}^{d}(S_{a}) \]
(no clause in \( P \) denies \( \omega \) from the specification)
THE DIAGNOSIS THEOREMS

Theorem 1

1. If there are no incorrect clauses, then the program is partially correct.
   \[ T^d_p(S_d) \leq S_d \]
   \[ S_d \text{ is a prefix of } T^d_p \]
   \[ F_d \frac{[P]}{S_d} \rightarrow d(\frac{[P]}{S_d}) \leq S_d \]

Theorem 3 does not hold for approximate observables.

Theorem 4 does not hold for approximate observables due to the absence of uncovered obstructions, even under the unique support assumption, does not imply program completeness.

Incompleteness bugs might be hidden by the approximation of the abstract semantics.

Theorem 6

If the program is partially correct, then

- If there exists an uncovered obstruction \( \rho \), then
  \( \rho \) is an incompleteness symptom.

Weaker results:

- Absence of incorrect clauses implies partial correctness.
- Uncovered obstructions correspond to bugs.
- Equivalent top-down characterization.
AN APPROXIMATE OBSERVABLE: DEPTH(k)

- The abstract domain of the observable \( T_k \)

- The concrete domain, where:
  - an equation \( x = t \) is replaced by an equation \( x = t' \)
  - every symbol of \( t \) at depth \( \geq k \) is replaced by a fresh variable

- The abstract immediate consequence operator

\[
T^k_p(I)(p(x_1, \ldots, x_n)) =
\bigcup \left\{ \varepsilon \mid p(x_1, \ldots, x_n) = t_1, \ldots, t_n \in \varepsilon \}, \exists \alpha \in q_{d}(x_1, \ldots, x_n) (I(\alpha) = \varepsilon_1), \exists \beta \in q_{m}(x_1, \ldots, x_n) (I(\beta) = \varepsilon_m) \right\}
\]

\[\varepsilon = T^k(\varepsilon \in \{ T_k(1 \leq \alpha \leq \varepsilon_1), \varepsilon_1 \rightarrow \varepsilon_m \}/x_1, x_n\}\]

\( g_{lb} \) is the one of the concrete domain

- \( T_k \) is an approximate observable \( \rightarrow \)
  - it is converging
  - \( T^k(\varepsilon \in [P]) \leq T^k_p \uparrow \omega = F^k_p ([P]) \)
AN EXAMPLE WITH \textsc{DEPTH(K)} - ANSWERS

\begin{verbatim}
accept([a|x]) :- acc(x).
accept([c|x]).
acc([b|y]) :- accept(x).
\end{verbatim}

1. Wrong version (missing clause) of an automaton which recognizes the language $L = \{(ab)^n | n \geq 0\} \cup \{(ab)^n | n \geq 0\}$

2. The specification of the intended \textsc{depth(2)}-answers

\[
S(\text{accept}(x)) = \begin{cases} \{x = [c], \{x = [a], \{x = [b, b]\}\}\} \\
\{x = [a, b], \{x = [b, y]\}\} \end{cases}
\]

\[
S(\text{acc}(x)) = \begin{cases} \{x = [c], \{x = [a], \{x = [b, b]\}\}\} \\
\{x = [a, b], \{x = [b, y]\}\} \end{cases}
\]

3. By applying the $T_p^{T_q}$ operator we find out that

- $P$ is partially correct w.r.t. $S$
- There exists an uncovered element

\[
\text{acc}(x) \Rightarrow \{x = [c]\}
\]

which shows that there is a missing clause for $\text{acc}$
**AN APPROXIMATE OBSERVABLE:**

**REPRESENTING GROUNDNESS BY POS**

- The abstract domain \((\text{POS}, \leq_{\text{POS}})\) [propositional formulas]
  
  (shown for two variables)

\[
\begin{array}{cccc}
\text{true} & \text{false} & \text{X} & \text{X} \leftrightarrow \text{Y} \\
\text{X} \land \text{Y} & \text{X} \lor \text{Y} & \text{X} \rightarrow \text{Y} & \text{X} \leftrightarrow \text{Y}
\end{array}
\]

\[
\begin{align*}
d_{\text{POS}} : \mathcal{P}(E) & \rightarrow \text{POS} \\
x_{\text{POS}} : \text{POS} & \rightarrow \mathcal{P}(E)
\end{align*}
\]

- \(\text{false} \rightarrow \emptyset\) (no answer)
- \(\text{true} \rightarrow \mathcal{P}(E)\) (no ground information)
- \(X \rightarrow \{E \mid \exists t \text{ grounded, } X = t \in E\}\) (X is grounded)
- \(X \land Y \rightarrow \{E \mid \exists t_{1}, t_{2} \text{ grounded}, \{X = t_{1}, Y = t_{2}\} \subseteq E\}\) (both X and Y are grounded)
- \(X \lor Y \rightarrow \{E \mid \exists t \text{ grounded, } X = t \in E \text{ or } Y = t \in E\}\) (either X or Y is grounded)
- \(X \rightarrow Y \rightarrow \{E \mid \exists t \text{ occurs in } t, X = t \in E\}\) (if X is grounded, then Y is pos)
- \(X \leftrightarrow Y \rightarrow \{E \mid \exists t, Y \text{ is the only variable in } t, X = t \in E\}\) (Y is grounded if Y is pos)

\((\text{POS}, \leq_{\text{POS}})\) is an approximate observable
A most general atom $p(x_1, ..., x_n)$

$$T^\text{pos}_p(I)(p(x_1, ..., x_n)) = \bigcup_{s \in \text{pos}} \{ s \}$$

$p(x_1, ..., x_n) : = e_1, ..., e_k \sqcup q_e(x_1^e, ..., x_{k1}^e), ..., q_m(x_1^m, ..., x_{km}^m)$ (renamed clauses of $p$)

$I(q_e(x_1^e, ..., x_{k1}^e)) = e_1,$

$\vdots$

$I(q_m(x_1^m, ..., x_{km}^m)) = e_m,$

$\varepsilon = \text{get}_{\text{pos}}\left(\{ \text{d}_{\text{pos}}(\{ e_2, ..., e_k \}), e_1, ..., e_m \} \right)$

- This is the only occurrence of "concrete constraints" in the definition
- It can be eliminated by considering the abstract program, where, in each clause,
  $$e_1, ..., e_k$$
  is replaced by
  $$\text{d}_{\text{pos}}(\{ e_1, ..., e_k \})$$

- Since $\text{pos}$ is an approximate observable, we know that it is condensing
- $\text{d}_{\text{pos}}(F_p[I_p]) \leq_{\text{pos}} T^\text{pos}_p \circ w = F_{\text{pos}}[I_p]$
\[ F[\mathbb{P}] (p(x)) = \{ \{ x = f(a) \} \} \]
\[ F[\mathbb{P}] (q(x)) = \{ \{ x = a \} \} \]
\[ F[\mathbb{P}] (r(x)) = \emptyset \]
\[ F[\mathbb{P}] (s(x,y)) = \{ \{ y = a \} \} \]

\[ d_{\text{pos}} (F[\mathbb{P}] (p(x)) = X \]
\[ d_{\text{pos}} (F[\mathbb{P}] (q(x)) = X \]
\[ d_{\text{pos}} (F[\mathbb{P}] (r(x)) = \text{false} \]
\[ d_{\text{pos}} (F[\mathbb{P}] (s(x,y)) = Y \]

The abstract semantics is not precise, i.e. \[ F_{\text{pos}} [\mathbb{P}] \neq \alpha_{\text{pos}} (F[\mathbb{P}]) \]
\[ f(x) := x = f(y) \land q(y). \]
\[ q(x) := x = a. \]
\[ r(x) := y = q(x) \land p(y). \]
\[ s(x, y) := \neg r(x). \]

\[
\begin{align*}
F(P)(p(x)) & = \begin{cases} X & \text{if } y = a \end{cases} \\
F(P)(q(x)) & = \begin{cases} X & \text{if } y = a \end{cases} \\
F(P)(r(x)) & = \emptyset \\
F(P)(s(x, y)) & = \begin{cases} Y & \text{if } y = a \end{cases}
\end{align*}
\]

\[ d_{pos} (F(P)(p(x))) = X \]
\[ d_{pos} (F(P)(q(x))) = X \]
\[ d_{pos} (F(P)(r(x))) = false \]
\[ d_{pos} (F(P)(s(x, y))) = Y \]

- The abstract semantics is not precise, i.e. \[ F_{pos}(P) \neq d_{pos}(F(P)) \]

\[
\begin{align*}
S_{pos}(p(x)) & = X \\
S_{pos}(q(x)) & = X \\
S_{pos}(r(x)) & = false \\
S_{pos}(s(x, y)) & = Y
\end{align*}
\]

\[
\begin{align*}
T_{pos}(S_{pos}(p(x))) & = X \\
T_{pos}(S_{pos}(q(x))) & = X \\
T_{pos}(S_{pos}(r(x))) & = X \\
T_{pos}(S_{pos}(s(x, y))) & = Y
\end{align*}
\]

- P is partially correct and complete.
- However, the third claim is incorrect on \( x = \neg r(x) \) (because of the approximation in the abstract \( T_P \) ).
Diagnosis and verification of partial correctness

- A specification like

\[ S(+ (x, y, z)) = x \land y \land z \]

reads as

"Every successful computation of ?-+ (x, y, z) binds x, y, and z to ground terms."

- It looks like an assertion (postcondition).

- Diagnosis can be compared to techniques for verifying partial correctness

(Apr 83, 84, 86)
WHAT IS AN ASSERTION?

- verification
  - an intensional set of concept atoms
    - closed under instantiation
      - the intensional definition is given in terms of concepts (being a list, being ground, ...)
        which have to be defined elsewhere (e.g. the theory of proposition)
        \[ \{ +(x, y, z) \mid x \text{ is ground} \land y \text{ is ground} \land z \text{ is ground} \} \]
        - infinitely many concrete atoms
  - diagnosis
    - an extensional set of abstract atoms
      \[ \{ +(x, y, z) : -x\land y \land z \}\]
      - one abstract atom
    - the constraint on being closed under instantiation may not have interesting properties (e.g. computed answers, freeness, ...)
    - in the case of precise observables (e.g. correct answers)
      intensional assertions can lead to finite assertions, but
      "no additional concepts, apart from the abstract ones, are needed in abstract diagnoses."

WHAT IS A SPECIFICATION?

- Verification
  - a pair of reactions
    - precondition and postcondition
      - the precondition specifies the class of goals we are interested in

- Diagnosis
  - one question
    - postcondition only
      - "pre-independent" verification, or verification for "most general atomic facts"
There exist two different formalizations:

- The first one corresponds to an notion of partial correctness.
  (as the actual outcome is included in the specification)
  (strongest postcondition)
- The second one corresponds to our notion of partial correctness and completeness.
  (as the actual outcome is the same as the specification)
Verification

- The definition of well-entailed (specification) programs w.r.t. a specification
  a simple inductive definition which generates a set of formulas which have to be proved in the theory which formalized the "external" concepts used in the operations

4) Diagnoses

- The application of the abstract Tp to the specification
  - Abstract computation instead of theorem proving.
How do we prove partial correctness?

- Partial correctness 1
  - Verification
    - Partial correctness is essentially proving the well-orderedness

- Objections
  - Absence of incorrect clauses

- Partial correctness 2 (partial correctness + completeness)
  - Verification
    - Requires the construction of a concrete semantics
      (least Herbrand model of a suitable program, obtained from P and the preconditions)

- Objections
  - Absence of unearned elements
    - If the observable is precise and the abstract Tp has one fixpoint only
    - If the observable is "precise for P" and the abstract Tp has one fixpoint only
WHAT IF THE PARTIAL CORRECTNESS PROOF FAILS?

**Verification**

- no useful information

**Diagnosis**

- incorrect elements always correspond to incompatence bugs
- incorrect elements correspond to incompatence bugs, if the domain is precise
**Verification**

- The ability to handle specifications given in terms of pre- and post-conditions

**Diagnosis**

- Generates information useful for debugging
- One step of abstract computation instead of theorem proving
- Possibly better for proving completeness
- Can use tools from abstract interpretation theory (domain composition, domain refinement) to improve the precision and the expressiveness of specifications

\[ 
\downarrow 
\]

Extension of diagnosis to handle pre and post-conditions
• The program is split into $m$

  predicate-disjoint modules
  
  • any predicate is fully defined by a single module
  • it is not a hierarchical decomposition
    (mutual recursion across modules is allowed)

• each module $P_i$ has a specification $I_{i,2}$

• $\text{use } (P_i) = \left\{ \begin{array}{l} I_{i,2} \quad \text{the body of a clause in } P_i \text{ contains a predicate} \\
\text{defined in module } P_j \end{array} \right\}$

• two different (strongly related) techniques

  1. (modular diagnosis)
    
    • all the program components are known
    • the diagnosis is performed component-wise

  2. (modular development and diagnosis)
    
    • we have one component only (and all its related specifications)
    • we perform the diagnosis of a single module
**MODULAR DIAGNOSIS AND COMPOSITIONALITY**

- Modular analysis is usually based on OR-compositional semantics.
  - For example, [Codish, Debray, Giacobazzi, POPL 83] use the OR-compositional version of the S-semantics.
  - [Bacci, Bicchi, Levi, Mes, TCS 84]

- Our semantics are not OR-compositional.

- Abstract diagnosis does not require to actually compute the abstract semantics.
  - Both top-down and bottom-up diagnoses just use the abstract immediate consequence operators $T_\pi$, which are indeed OR-compositional.

- We do not need any modification in the basic framework.
\[ b = 6 \quad u \quad 6 \]
\[ I = \overline{I} \quad u \quad \overline{I} \]
\[ a \in \mathbb{F}[g(f)] \Rightarrow f \]
\[ a \in \mathbb{F}[g(f)] \]

**Consequence**

\[ \text{If } \theta' \in (I, g(f)) \text{ and } a \neq T_{g(f)}(I, \theta') \text{ then } a \notin T_{g(f)}(I, \theta') \]

**Corollary**

If \( \theta' \in (I, g(f)) \) then \( a \notin T_{g(f)}(I, \theta') \)

**Note**

- If \( \theta' \) is a universal, then \( a \notin T_{g(f)}(I, \theta') \)
we want to perform the diagnosis of \( P_i \) under the assumption that all the other (possibly not yet implemented) modules are correct and complete (w.r.t. their specifications).

- We cannot construct the concrete semantics of \( P_i \) since the other components are unknown.

- We can take as concrete semantics of the unknown modules the formalization of their abstract specifications.

\[
T_{P_i}^{I_0} (I) = T_{P_i} (I \cup \text{use}(P_i))
\]

\[
F^{I_0} [P_i] = T_{P_i}^{I_0} \cdot W
\]

module correctness\hspace{1cm} d(F^{I_0} [P_i]) \leq d \cdot I_0^i

module completeness\hspace{1cm} I_0^i \leq d (F^{I_0} [P_i])

the results

\[ P = P_1 \cup \ldots \cup P_m \]

- If all the \( P_i \)'s are correct (complete), then \( P \) is correct (complete).
- If there are no m-incorrect elements in \( P_i \), then \( P_i \) is m-correct.
- If \( P_i \) and \( d \) are complete, then if there exists an m-incorrect element in \( P_i \), \( P_i \) is not m-correct.
- If \( P_i \) is correct, and there exists an m-incorrect element in \( P_i \), \( P_i \) is not m-complete.
- If \( d \) is complete (and we assume \( T_{P_i} \)'s have a unique fixed point), then if there are no m-incorrect elements in \( P_i \), \( P_i \) is m-complete.