most interesting properties of logic programs are properties of computed answers

- groundness
  - essential component of normal order analysis
    - independence
    - sharing
    - occur-check, ...
  - if the result of the analysis tells us that $X$ is ground
    - the constraint guarantees that every possible execution will bind $X$ to a ground term

- linearity
  - sharing
  - $\{ 1, 2, \ldots, 1 \times y, z, \ldots \}$
    - $X, Y, Z$ may be bound to terms containing the same variable

- types
  - un ejemplo parlando di verificare

- we will look at "groundness" abstract domain
rather than defining a semi-project or non-denotational observable modeling a suitable groundness domain (from SLD-derivations) we take as concrete instance a semantics modeling up computed answers

- a denotational observable
- we will focus on the denotational semantics only (just to make the presentation simpler)

- the denotational definition of an "equational version" of the S-semantics
- we will omitted the concrete domain only $\mathcal{P}(\mathcal{E}pms)$

- it is equivalent to defining a (semi-denotational) observable which is the composition of abstract (procedures) computed answers

\[ \text{abstract (procedures) computed answers} \]
\[ C \left[ \mathit{p}(\mathcal{F}): -e, q_1(\overline{f_1}), \ldots, q_m(\overline{f_m}) \right]_I = \{ \langle p(\overline{x}), e \rangle \mid \exists \langle q_i(x_i), E_i \rangle \in I, \]
\[ e_i \in E_i, \]
\[ e' = (\overline{x} = \overline{e} \land e \land \overline{x_1} = E_2 \land \ldots \land \overline{x_m} = E_m) \]
\[ e' \text{ is satisfiable}, \]
\[ e' \in E' \} \]

\[ \mathcal{P}\left[ \text{emptyproc} \right]_I = \{ \langle p(\overline{x}), \phi \rangle \mid p \in \mathcal{P} \} \]

\[ \mathcal{P} \left[ \mathit{c; P} \right]_I = \{ \langle p(\overline{x}), e \rangle \mid \langle p(\overline{x}), e_2 \rangle \in C \left[ \mathit{c} \right]_I, \]
\[ \langle p(\overline{x}), e' \rangle \in \mathcal{P} \left[ \mathit{P} \right]_I, \]
\[ E = E_2 \cup E_2 \} \]

- concrete domain
- \( \mathcal{P}(\mathcal{E}p) \)
- \( e \in \mathcal{E}p \), \( E \in \mathcal{P}(\mathcal{E}p) \)

The partial order is commutativity \( \leq \)
assume \( d \) is an abstraction function from \( P(Eqs) \) to a given abstract domain

\[
d^d(p(X)) = \{ <p(x), F> | \exists <p(x), F> \in E \}
\]

The abstract semantic interpretation is:

\[
d^d(p(X)) = \{ <p(x), F> | \exists <p(x), F> \in E \}
\]

The abstract semantic equations are obtained from the concrete one by replacing \( \land \) and \( \lor \) by \( \text{geb} \) and \( \text{lub} \) respectively.

\[
C^d \left[ p(t) \right]_{Id} = \left\{ <p(x), t> | p \in P \right\}
\]

\[
P^d \left[ C ; P \right]_{Id} = \left\{ <p(x), F> | \exists r(x) \in C \in S^d \left[ C \right]_{Id}, \exists q(x) \in P \in S^d \left[ P \right]_{Id}, F = \text{lub}(\{r(x), f\}) \right\}
\]

\[
F^d(p) = \text{geb}(\{r(x), f\}) \uparrow w
\]
THE FIRST GROUNDNESS

ABSTRACT DOMAIN

- concrete domain \((P(Eqns), \subseteq)\)
  sets of sets of equations in sorted form

- abstract domain \((\text{ground}, \subseteq)\)
  Shown in 2 variables

\[
\begin{array}{c}
\text{true} \\
X \\
\text{false} \\
Y
\end{array}
\]

- concretization function

\[
\delta(x) = \begin{cases} 
\text{Eqns}, & \text{if } x \text{ a true} \\
\{ e | x_i = c_i \text{ ce }, c_i \text{ ground} \}, & \text{if } x = x_i \wedge \neg x_i
\end{cases}
\]

\(X\) represents the set of all sets of equations where \(X\) is bound to a ground term

- abstraction function

\[
d(y) = \begin{cases} 
\text{true}, & \text{if } y \subseteq \text{Eqns} \\
\neg x_i \wedge \neg x_i, & \text{if } \text{in every } c_i \text{ a true}\, x_i = 5 \text{ or } c_i, \text{ ce ground} \\
x_i = c_i \text{ or } c_i, \text{ ce ground}
\end{cases}
\]
$E_1 = \{ \{ x = f(y, z), \ w = a \} \}$

$\text{ground}(E_1) = \emptyset$

$E_2 = \{ \{ x = a \}, \{ x = f(y) \} \}$

$\text{ground}(E_2) = \text{true}$

$E_3 = \{ \{ x = a \}, \{ y = a \} \}$

$\text{ground}(E_3) = \text{true}$
AN ABSTRACT SEMANTICS

\[
p: p(a, v), p(x, b), q(x, y).
\]

- The (concrete) \(S\)-semantics

\[
\mathcal{F}[P] = \begin{cases} 
< p(x, y), \{ \{ x=a \}, \{ y=b \} \} >, \\
< q(x, y), \{ \{ x=y \} \} >, \\
< r(x, y), \{ \{ x=a, y=a \}, \{ x=b, y=b \} \} > \end{cases}
\]

- Observing prowemon on the concrete semantics

\[
d_{\text{ground}}(\mathcal{F}[P]) = \begin{cases} 
< p(x, y), \text{true}, >, \\
< q(x, y), \text{true}, >, \\
< r(x, y), x \wedge y > \end{cases}
\]

- Computing the abstract semantics

\[
P_{\text{ground}}[P] \uparrow 1 = \begin{cases} 
< p(x, y), \text{true}, >, \\
< q(x, y), \text{true}, > \end{cases}
\]

\[
P_{\text{up}}[P] = P_{\text{ground}}[P] \uparrow 2 = \begin{cases} 
< p(x, y), \text{true}, >, \\
< q(x, y), \text{true}, >, \\
< r(x, y), \text{true}, > \end{cases}
\]

\[f_{\text{up}}[P] \text{ is less precise than } d(\mathcal{F}[P])\]
A BETTER DOMAIN FOR GROUNDNESS ANALYSIS: GROUND DEPENDENCIES

- Concrete domain \( (\mathcal{P}(\text{Eqns}), \subseteq) \)
- Abstract domain \( (\text{Def}, \subseteq) \)

shown for 2 variables

\[
\begin{align*}
\text{X} & \rightarrow \text{Y} \\
\text{X} & \iff \text{Y} \\
\text{x} & \rightarrow \text{y} \\
\text{x} & \land \text{y}
\end{align*}
\]

- Concatenation function

\[
g(x) = \begin{cases} 
\text{Eqns}, & \text{if } x = \text{true} \\
\{ e \mid x = \exists i \in e, \text{bi-variant} \}, & \text{if } x = \text{X} \land \ldots \land \text{Xn} \\
\{ e \mid x = \exists i \in e, X_{\text{eq}}(i) \}, & \text{if } x = \text{X} \rightarrow \text{Y} \\
\{ e \mid x = \exists i \in e, \text{Y}_{\text{eq}}(i) \}, & \text{if } x = \text{X} \rightarrow \text{Y}_{\text{eq}}
\end{cases}
\]

\text{X} \land \text{Y}  \quad \text{all the sets of equations where } \text{X} \text{ and } \text{Y} \text{ are bound to ground terms}

\text{X} \rightarrow \text{Y}  \quad \text{all the sets of equations which contain the equation } \text{X} = \text{E}, \text{such that } \text{Y} \text{ occurs at } \text{E}  \\
\text{(if } \text{X} \text{ is ground then } \text{Y} \text{ is ground too)}

\text{X} \rightarrow \text{Y}_{\text{eq}} \quad \text{all the sets of equations which contain } \\
\text{X} = \text{E}, \text{such that } \text{Y}_1 \text{ and } \text{Y}_2 \text{ are the only variables occurring in } \text{E} \\
\text{(if } \text{Y} \text{ is ground then } \text{Y}_1 \text{ and } \text{Y}_2 \text{ are ground and variables)}
\[ E_1 = \{ \{ x = f(y, z), w = a \} \} \]

\[
\begin{align*}
\sigma_{\text{ground}}(E_1) &= w \\
\sigma_{\text{def}}(E_1) &= w \land (x \equiv y \land z) \land (x \rightarrow y) \land (x \rightarrow z)
\end{align*}
\]

\[ E_2 = \{ \{ x = a \}, \{ x = f(y) \} \} \]

\[
\begin{align*}
\sigma_{\text{ground}}(E_2) &= \text{true} \\
\sigma_{\text{def}}(E_2) &= x \equiv y
\end{align*}
\]

\[ E_3 = \{ \{ x = a \}, \{ y = a \} \} \]

\[
\begin{align*}
\sigma_{\text{ground}}(E_3) &= \text{true} \\
\sigma_{\text{def}}(E_3) &= \text{true}
\end{align*}
\]
THE ABSTRACT SEMANTICS ON DEF

\[ p(a, y). \]
\[ p(x, b). \]
\[ q(x, x). \]
\[ f(x, y) := -p(x, y), q(x, y). \]

- The (concrete) s-semantics

\[ f[[p]] = \{ p(x, y), \{ x = a \}, \{ y = b \} \}, \]
\[ p(x, y), \{ x = a \}, \{ y = b \} \}
\[ q(x, y), \{ x = a \}, \{ y = b \} \} \}

- Defining DEF-promotion on the conrcte semantics

\[ d_{def}(f[[p]]) = \{ p(x, y), \text{true}, \]
\[ q(x, y), x \leftrightarrow y, \]
\[ r(x, y), x \wedge y \} \]

- Computing the abstract semantics

\[ \Theta_{def}[[p]] \uparrow 1 = \{ p(x, y), \text{true}, \]
\[ q(x, y), x \leftrightarrow y \} \]
\[ \Theta_{def}[[p]] \uparrow 2 = \{ p(x, y), \text{true}, \]
\[ q(x, y), x \leftrightarrow y \} \]
\[ r(x, y), x \wedge y \} \]

- The abstract semantics is still less precise than the abstraction of the concrete semantics, yet it is better than the concrete.
Improving the abstract domain for groundedness analysis:
Some disjunctive information

Concrete domain \((\mathcal{P}(\text{Epns}), \leq)\)

Abstract domain \((\text{Pos}, \leq)\)

shown for two variables

\[
\begin{align*}
X &\rightarrow Y \\
X &\leq Y \\
X &\leq Y \\
X &\leq Y \\
Y &
\end{align*}
\]

congruence function

We are defined for def plus the new case:

\[
\begin{align*}
x &\leq e, e \leq e, \text{ground or} \\
y &\leq b \leq e, b \leq e \text{ground?} \text{, if } x \leq x \land y
\end{align*}
\]

\(x \land y\) represents all the sets of equations when either \(x\) or \(y\) are bound to a ground term.
\[ E_1 = \{ \{ x = f(y, z), w = a \} \} \]

\[ \begin{align*}
\text{ground}(E_1) &= w \\
\text{def}(E_1) &= w \land (x \leftrightarrow y \land z) \land (x \rightarrow y) \land (x \rightarrow z) \\
\text{pos}(E_1) &= w \land (x \leftrightarrow y \land z) \land (x \rightarrow y) \land (x \rightarrow z)
\end{align*} \]

\[ E_2 = \{ \{ x = a \}, \{ x = f(y) \} \} \]

\[ \begin{align*}
\text{ground}(E_2) &= \text{true} \\
\text{def}(E_2) &= x \leftrightarrow y \\
\text{pos}(E_2) &= x \lor (x \leftrightarrow y)
\end{align*} \]

\[ E_3 = \{ \{ x = a \}, \{ y = a \} \} \]

\[ \begin{align*}
\text{ground}(E_3) &= \text{true} \\
\text{def}(E_3) &= \text{true} \\
\text{pos}(E_3) &= x \lor y
\end{align*} \]
The abstract semantics on POS

\[ P:\]
\[ p(a, y).
\[ p(x, b).
\[ q(x, x).
\[ r(x, y) \equiv p(x, y), q(x, y). \]

The (concrete) s-semantics
\[ \mathcal{F}([P]) = \{ <p(x, y), x = a, y = b> \}
\[ <q(x, y), x = y> \}
\[ <r(x, y), x = a, y = b> \} \]

Observing Pos-groundness on the concrete semantics
\[ d_{\text{pos}}(\mathcal{F}([P])) = \{ <p(x, y), x \neq y> \}
\[ <q(x, y), x = y> \}
\[ <r(x, y), x \land y> \} \]

Computing the abstract semantics
\[ \mathcal{P}_{\text{pos}}^{d_{\text{pos}}([P])} \uparrow 1 = \{ <p(x, y), x \neq y> \}
\[ \mathcal{P}_{\text{pos}}^{d_{\text{pos}}([P])} \uparrow 2 = \]
\[ \{ <p(x, y), x \neq y>,
\[ <q(x, y), x = y>,
\[ <r(x, y), x \land y> \} \]

The abstract semantics is more precise than \( \mathcal{F}([P]) \) and, on this example, as precise as the abstraction of the concrete semantics.

\[ g_{\text{rel}}(x = a, x \neq y) \]
- the property (formalism) possibly non-ground

- conjunctive completion (to get a horn formula)

- Heyting completion (functional dependence)

  \[ \text{Def} = \text{Ground} \rightarrow \text{Ground} \]

- awoken Heyting completion (looks like a disjunction)

  \[ \text{Pos} = \text{Def} \rightarrow \text{Def} \]

  \[ \text{Pos} = \text{Pos} \rightarrow \text{Pos} \]

  - improvement in precision
  - some unprofitability properties may be verified

  The same construction can be applied to other domains

  - types
. The abstract domain is indeed an abstraction of \( P(\text{Expr}) \) and the abstract semantics is an abstraction of the s-structures.

. If one wants to analyze precision of computed answers.

. If we want to know the bounded information for procedure cells (to optimize the compiled code), we have to combine:
  - Call patterns
  - Boundness abstraction \( \rightarrow \) semi-denotational observable?

. If we want a modular boundness analysis, we need to combine:
  - An \( \alpha \)-compositional semantics (SCD-annotations, included)
  - Boundness abstraction \( \rightarrow \) semi-binding observable?

. If we want a top-down boundness analysis, we need to combine:
  - A perfect observable
  - Boundness abstraction \( \rightarrow \) superperfect observable?
**TOP-DOWN vs. BOTTOM-UP**

- **Top-down (operational) analysis**
  - We abstract the operational semantics (transition system)
  - Composition of an abstraction on substitutions and an abstraction on the structure of sub-expressions, corresponding to a perfect observable
    - SLD unification
    - results
  - More abstract observables, such as computed answers, lead to improved computations.

- **Bottom-up (denotational) analysis**
  - We abstract the denotational semantics (in particular, the suitable top-down)
  - Composition of an abstraction on substitutions and an abstraction on SLD-resolution, corresponding at least to a denotational semantics
    - computed answers
    - correct answers
    - ground answers of correct answers
    - safe answers
    - partial answers

  The bottom-up analysis is always performed by first analyzing the program (without the goal) and then jointly deriving the behaviors of the goal from the abstract semantics of the program.
goal-independence

- the analysis is of the abstract semantics of the program
  
  the least fixed point of the (denotational) abstract
  top operator

- the collection of (sometimes) behaviours for
  most general atomic goals

goal-dependence

- we only analyze the abstract behavior of a
  specific goal

  - we apply the abstract behavior system to the
    (abstract) goal

  - then using denotational bottom-up methods the
    behavior of a goal is always derived from the
    abstract semantics of the program (goal
    concomitantly by construction)

goal-dependent (top-down) analyses may sometimes give
more precise results than goal-independent (top-down or
bottom-up) analyses

- goal-independent computations are as precise as
  goal-dependent ones, if the obtained is concomitent,
  i.e., if the abstract behavior derived from the goal
  independent abstract semantics is the same as the one
  that would be (top-down) computed
  for the specific goal
Relation Among Abstract Semantics

- top-down (goal-independent) program denotation
  \( O_d[p] \)

- bottom-up (goal-independent) program denotation
  \( F_d[p] = P_d[p] \uparrow_w \)

- goal-dependent top-down program semantics
  \( B_d[G \downarrow p] \)

- denotational semantics of a goal

- in the concrete semantics
  \( O[p] = F[p] = P_p[p] \uparrow_w \)
  (equivalence of top-down and bottom-up denotation)

- perfect observables
  - equivalence to denotation
  - top-down / bottom-up
  - goal-dependent / goal-independent
  - completeness of abstract denotational semantics

- denotational observables
  - completeness of abstract denotational semantics

- semi perfect observables
  - top-down = bottom-up
  - goal-dependent = goal-independent

- semi denotational observables
- We want to compute "operationally" computed answers (a denotational observable).
- We want to compute "operationally" computed answers abstractions to elements of Pos (a semi-denotational observable).

- The abstract-humam-form systems are too imprecise.
  - We cannot perform the abstraction at every computation step.
  - We can choose a (more concrete) perfect (semi-perfect) observable, compute on that domain and then (at the end) perform the abstraction.

- How we compute answers by computing selfextensions or reictions.
- How we compute from other information for answers by computing or "abstract" self-extensions or reictions.

- Occasionally in order to do something operationally we sometimes need to be more concrete than in the denotational case.
how to do "top-down" operational abstract computations

- SLD-derivations, where substitutions (ganzteils) are replaced by formulas in Ground, Act or PAs

- We can check whether the various domains are "enduring," i.e. whether a substitutability can:
  - This pool-dependent behavior can be derived without looking precision from the pool-independent deduction
  - The abstract denominational corresponds to a (usually more abstract manageable) is also equivalent

- Ground and Def are not enduring, while PAs is.
Def is not condensing.

\[ p(a, 1), \]
\[ p(x, b), \]
\[ q(x, x). \]

\[ P \]

- goal-independent top-down abstract semantics

\[ C_d [P] = \{ <p(x, y), \text{true}>, <q(x, y), x \leftrightarrow y> \} \]

\[ \text{lab}(x, y) \]

- goal-dependent top-down abstract semantics

\[ B_d [\text{?-}p(x, y), p(x, y) \text{ on } P] = \]

\[ \text{?-}p(x, y), p(x, y). \rightarrow x \land y \text{ (lab ... )} \]

\[ X \]
\[ Y \]
\[ \text{?-}q(x, y) \]
\[ x \leftrightarrow y \]
\[ \text{get}(\{x, x \leftrightarrow y\}) \]

\[ \text{X \land Y} \]

\[ \text{?-}q(x, y) \]
\[ x \leftrightarrow y \]
\[ \text{get}(\{y, x \leftrightarrow y\}) \]

\[ \text{X \land Y} \]

- elaboration of the abstract goal semantics from \( C_d [P] \)

\[ \text{get}(\{\text{true}, x \leftrightarrow y\}) = x \leftrightarrow y \geq x \land y \]

(For some result would have been obtained by the elaboration construction)

- Pas wants correctly in this example and is indeed correct.
The abstraction from $\mathcal{P}(E_{\text{unseq}})$ to the abstract domain, takes place in the semantics of a clause:

$$
\mathcal{C}^a \left[ \left( p(C) \right) ; e, q_1(f_1), \ldots, q_n(f_n) \right]_{x} =
\begin{cases}
\exists q_i(x_i), F_i \in I^x, \\
F = \text{gb}(\{ d(x=E), d(c), d(x_1=E_1), \ldots, d(x_n=E_n), F_0, \ldots, F_n \})_{x, x_i}
\end{cases}
$$

The semantics of composition of clauses:

$$
\mathcal{P}^a \left[ C ; P \right]_{x} = \{ p(x), F \} \ | \ \exists p(x), F_2 \in \mathcal{C}^a \left[ C \right]_{x}, \\
\exists p(x), F_2 \in \mathcal{P}^a \left[ P \right]_{x}, \\
F = \text{sub}(\{ F_2, F_3 \})
$$

Rather than having the abstraction at every application of the semantic evaluation function, we can abstract the set of (enclose) equations in the program text, obtaining an "abstract program."

The abstract semantics of the abstract program is the (regular) semantics (over a different domain) of the abstract program.

More efficient, since we perform the abstraction once and for all at "compiled time." (Translation time)
2. Transformations

1. From logic program to equational CLP program

\[
\begin{align*}
  P(a, y) &= \text{} \\
  P(x, b) &= \text{} \\
  Q(x, x) &= \text{} \\
  R(x, y) &= -b(x, y), q(x, y)
\end{align*}
\]

\[
\begin{align*}
  P(x, y) &= -x = a \\
  P(x, y) &= -y = b \\
  Q(x, y) &= -x = y \\
  R(x, y) &= -p(x, y) \land q(x, y)
\end{align*}
\]

2. From equational CLP program to abstract program

\[
\begin{align*}
  P(x, y) &= -d(\{x = a\}) \\
  P(x, y) &= -d(\{y = b\}) \\
  Q(x, y) &= -d(\{x = y\}) \\
  R(x, y) &= -p(x, y) \land p(x, y)
\end{align*}
\]
\[ f(x_1) = c_1, q_2(x_2), \ldots, q_n(x_n) \]\(\forall x \in \mathcal{D} \)

\[
\{ p(x') \mid p(x), \alpha \prec q_i(x_i), \beta \in \mathcal{D} \text{ (suitably renamed)} \}
\]

- no more abstractions of concrete values
- gets only lub's of abstract values coming from the (abstract) program and the current approximation of the semantics
\[ \begin{align*}
\text{\texttt{p}}(x, y) & : = \texttt{a}(\{x = a\}) \\
\text{\texttt{p}}(x, y) & : = \texttt{a}(\{y = b\}) \\
\text{\texttt{g}}(x, y) & : = \texttt{a}(\{x = y\}) \\
\text{\texttt{r}}(x, y) & : = \texttt{p}(x, y) \land \texttt{g}(x, y) \\
\text{\texttt{b}}(x, y) & : = x \\
\text{\texttt{p}}(x, y) & : = y \\
\text{\texttt{g}}(x, y) & : = x \leftrightarrow y \\
\text{\texttt{r}}(x, y) & : = \texttt{p}(x, y) \land \texttt{g}(x, y) \\
\text{\texttt{true}} & \\
\langle \text{\texttt{p}}(x, y), \text{\texttt{emb}}\{x, y\} \rangle \\
\langle \text{\texttt{g}}(x, y), x \leftrightarrow y \rangle \\
\langle \text{\texttt{r}}(x, y), \text{\texttt{geb}}\{\text{\texttt{true}}, x \leftrightarrow y\} \rangle \\
\# \\
\langle x \leftrightarrow y \rangle
\end{align*} \]
AN EXAMPLE ON DEF

\[ \alpha(x, y, z) := x = [1], y = 2 \]
\[ \alpha(x, y, z) := x = [x_1 x_2], z = [x_1 z_2], a(x_2, y, z) \]

\[ a(x, y, z) := x \land y \Rightarrow z \]
\[ a(x, y, z) := x \Rightarrow x_1 \land x_2, z \Leftrightarrow x_1 \land z_2, a(x_2, y, z) \]

\[ \mathbb{P}^d[\alpha] \uparrow_0 = \{ \langle a(x, y, z), 1 \rangle \} \]
\[ \mathbb{P}^d[\alpha] \uparrow_1 = \{ \langle a(x, y, z), \text{LUB}(\{ x \land y \Rightarrow z \}) \rangle \}
\[ \text{GLB}(\{ x_1 \land x_2, y \Rightarrow x_1 \land x_2 \land z_2 \}) \}
\]
\[ Z \Leftrightarrow (x \land y) \]

- If we make another iteration we generate the same "interpretation" which is therefore the fixpoint.