Horn Clause Logic

A generic clause

\[ A_1 \lor \ldots \lor A_k \lor \neg B_1 \lor \ldots \lor \neg B_m \]

can equivalently be represented in the form

\[ A_1, \ldots, A_k \rightarrow B_1, \ldots, B_m \]

conclusions hypotheses

to be "logically read" as

\[ (B_1 \lor \ldots \lor B_m) \lor (A_1 \lor \ldots \lor A_k) \]

Horn clauses are clauses with at most one conclusion (\( k \leq 1 \))

- definite (program) clauses \( k = 1 \)
  \[ A \rightarrow B_1, \ldots, B_m \]

- unit clause (assertion) \( k = 1, m = 0 \)
  \[ A \rightarrow \]

- goal (negative) clauses, queries \( k = 0 \)
  \[ \neg A \rightarrow B_1, \ldots, B_m \]

- empty clause \( k = 0, m = 0 \)
- A program is a set of definite clauses.

\[ \forall x_1, \ldots, x_n \ (B_1 \lor \ldots \lor B_n) \]

The goal is \( B_1 = B_n \)

- Formulas which can be proved are restricted to Horn clauses or formulas restricted to definite Horn clauses.

  - Horn clause theories are always consistent.
  - There exists a complete resolution strategy which is exhaustively non-deterministic.
  - There exists a canonical Herbrand model.
  - A goal refutation returns a substitution (computed answer)
Herbrand Universe
the set of all the ground terms

Herbrand Base

- if the language has no constant symbols, we add an arbitrary constant symbol

Herbrand Interpretation

- the domain is the Herbrand Universe
- constants, functions and predicates are interpreted "syntactically"
- a Herbrand interpretation can be represented as any subset of the Herbrand Base (the set of ground atoms which are true!)

Herbrand Model

- Herbrand Interpretation must make all the clauses in C true in I
The logic program

\[ N = \neg A_1, \ldots, \neg A_n \text{ goal} \]

and clauses in \( P \)

\[ C_1 = \neg A_1 \rightarrow B_1, \ldots, B_m \text{ clause in } P \]

If for some \( i \), \( A_i \) and \( \neg B_i \) are unifiable with \( v \):

the new goal is

\[ N' = \neg (A_1, \ldots, A_{i-1}, B_i, B_{i+1}, \ldots, B_m, A_{i+1}, \ldots, A_n) \]

the resolvent of \( N \) and \( C \)

- Selection of the atom \( A_i \) in the pool (selection rule)
- Unification of \( A_i \) with some head \( A \rightarrow v \) (if possible)
- Replacement of preceding \( A_i \) with
  - preceding body \( B_i, \ldots, B_m \)
- Application of \( v \) to the resulting clause (proceeding
  - in both directions)

SLD - derivation of \( P \cup \{N'\} \)

- A sequence of goals \( \neg N_0, \neg N_1, \ldots \)
- A sequence of varant of clauses in \( P \)
  \[ C_0, C_1, \ldots \]
- A sequence of substitutions \( \delta_0, \delta_1, \ldots \)

- \( N_i \) is a resolvent of \( N_i, C_i \) with \( v \) of \( C_i \)
- \( C_i \) does not share variables with \( \neg N_0, C_0, \ldots, C_{i-1} \)

- If \( N_i = \square \), the derivation terminates and is called
  - an SLD - refutation
The policy chosen to select the atom in the goal may depend on the history of the derivation. We will almost always assume the leftmost selection rule (the rule of PROLOG).

SLD-TREE

Given a selection rule, we can still have nondeterminism in the construction of an SLD-derivation, which (variant of) clause is chosen where head unifies with the selected atom.

The set of all possible SLD-derivations (for a given selection rule) are represented by an SLD-tree.

- The root is N.
- Each node is a goal.
- The members of a node are all its resolvents with variants of clauses of P, which heads unify with the selected literal.

The search rule specifies how to visit the SLD-tree. Whenever necessary, we assume a "fair" search rule.
topic program

\[ P \]

\[ N = \leftarrow A_1, \ldots, A_k \]

- Let us assume there exists an SLD-refutation of \( P \cup N \), whose sequence of substitutions is \( \sigma_0, \ldots, \sigma_m \).

- The general correctness result for the resolution method tells us that

\[ \exists X_1, \ldots, X_n (A_1 \land \ldots \land A_k) \vdash \sigma_0 \vdash \ldots \vdash \sigma_m \]

is logical consequence of \( P \).

- A stronger result holds: the universal closure of the formula

\[ A_1 \land \ldots \land A_k \vdash \sigma_0 \vdash \ldots \vdash \sigma_m \]

is logical consequence of \( P \).

\[ \text{COMPUTED ANSWER SUBSTITUTION} \]

- The restriction to the variables occurring in \( N \) of

the composition of substitutions \( \sigma_0 \vdash \ldots \vdash \sigma_m \)

- The stronger correctness result gives a semantic

(model-theoretic) meaning to computed answers.
The proof-theoretic program


sometimes called "operational semantics"

- program P
- Herbrand box B_P

O_P = \{ A \in B_P | \text{PU} \vdash A \text{ has an SLD-derivation} \}

- subset set
- set of usable ground atoms

- it is a Herbrand interpretation
- is it also a model?
- The set of Herbrand interpretations is partially ordered by set inclusion in a complete lattice.

- A continuous operator from Herbrand interpretations to Herbrand interpretations:
  
  \[ T_D : 2^{BP} \rightarrow 2^{BP} \]

  \[ T_P(I) = \left\{ A \in BP \middle| \begin{array}{l}
    \exists a \in A_1, \ldots, A_n \text{ a ground instance of a clause in } P \\
    \land (A_1, \ldots, A_n) \subseteq I
  \end{array} \right\} \]

  (immediate consequences operator)

- Herbrand models of \( P \) are pre-fixpoints of \( T_P \)

- Every logic program \( P \) has a Herbrand model \( H_P \) with the following properties:
  - \( H_P \) is the least Herbrand model of \( P \)
  - \( H_P \) is the least fixpoint of \( T_P \)
  - \( H_P = T_P H_P \)
  - \( H_P \) is the intersection of all the Herbrand models
  - \( H_P \) is the set of all the ground atoms within an logical consequence of \( P \)
  - \( H_P = \emptyset \)

---

**THE MODEL-THEORETIC PROGRAM DENOTATION**

- \( H_P \) the least Herbrand model of \( P \)
- \( H_P \) the least fixpoint of \( T_P \) (fixpoint or denotational semantics)
• correct answer substitution

\[ P \text{ program} \]
\[ N = a_1 \land \ldots \land a_n \text{ goal} \]

\( \phi \) is a correct answer substitution for \( P \cup \{ N \} \) if

- domain(\( \phi \)) contains only variables occurring in \( N \)
- \( \forall ( A_1 \land \ldots \land A_n) \phi \) is a logical consequence of \( P \)

- (as a corollary of the correctness theorem)

every computed answer substitution is a correct answer substitution

• Completeness theorem (Coven, 1979)

\[ P \text{ program} \]
\[ N \text{ goal} \]

if \( \phi \) is a correct answer substitution for \( P \cup \{ N \} \),
then exists a computed answer substitution \( \phi' \) for \( P \cup \{ N \} \)
such that \( N \phi' \) is an instance of \( N \phi \)

for every correct answer substitution there exists a "more generic" computed answer substitution

- also weaker consequence

if \( P \cup \{ N \} \) is inconsistent, there exists an SLD-refutation

• independence from the selection rule

computed answers do not depend upon the selection rule

- other properties (such as finite failures) do depend
THE S-SEMANTICS APPROACH

A survey with extensive list of references


Inadequacy of the standard declarative semantics

- $M_p = O_p = F_p$
  - least Herbrand model
  - Scott model
  - fixed-point semantics
  - one of the most salient properties of logic programs

- The declarative semantics (logic demodulation) does not capture relevant computational properties, such as computed answers

- The correctness and completeness theorems related to correct and computed answers are stronger than the equivalence theorem $O_p = M_p$

- Computed answers are just what makes Horn clause logic a programming language

- We need to look at logic languages as "standard" programming languages
the answer depends on

- what do we need the semantics for?
  - specification for the language implementation
  - to allow the user to understand the meaning of his/her programs
  - as basic semantics for reasoner-based tools
    (analysis, verification, transformation, interpretation and compiler generation, ...)

- Which computational properties we want to model (observables)?
  - success termination
  - computed answers
    - intermediate (partial) computed answers
    - procedure calls (call patterns)
  - SLD-trees

- Some observables are clearly more abstract than others
- Some observables are more adequate to a specific use of the semantics

Conclusion

There exists no such a thing as the semantics
The starting point (the most concrete semantics) is the proof-theoretic one, i.e., SLD-trees.

An observable α is any property which can be observed in an SLD-tree (a formal definition later).

- SLD-trees, SLD-derivations, rewrites,
- call patterns, partial answers, computed answers,
- finite failures, etc.

The choice of the observable α induces an equivalence relation on programs.

\[ P_1 \equiv_\alpha P_2 \iff P_1 \text{ and } P_2 \text{ are not distinguishable by any observation} \]

\[ W^\alpha_P = \{ \langle G, \sigma \rangle \mid \text{or } \alpha \text{ is the value of the observable } \alpha \text{ in the SLD-tree of goal } G \} \]

\[ P_1 \equiv_\alpha P_2 \iff W^\alpha_{P_1} = W^\alpha_{P_2} \]

For every goal \( G \), the observations in \( P_1 \) and \( P_2 \) are the same.

**Example**

The observable "success" \( \alpha_d \)

\[ P_1 \equiv_{\alpha_d} P_2 \iff \forall \text{ goal } G. \sigma^P_{\alpha_d}(G) = \sigma^P_{\alpha_d}(G) \]

The value of the observable \( \alpha_d \)

\[ \sigma^P_{\alpha_d}(G) = \begin{cases} \text{yes, if } G \rightarrow^* \text{ true} \\ \text{no, otherwise} \end{cases} \]
observables and denotations

- A program
  \[[P]\] \text{ a denotation of } P \text{ ("semantics")}

- The denotation is correct w.r.t. the observable \( \alpha \), if
  \[[P_1]\] = \[[P_2]\] \implies P_1 \cong_\alpha P_2 \quad \perp P_1, P_2

- An essential property:
  a semantics which identifies programs which have a different behavior w.r.t. \( \alpha \) is useless if one wants to reason about the observable \( \alpha \)

- The denotation is minimal w.r.t. the observable \( \alpha \), if
  \( P_1 \cong_\alpha P_2 \implies [[P_1]] = [[P_2]] \quad \perp P_1, P_2

- If a denotation is correct and minimal w.r.t. \( \alpha \), then the equivalence relation induced by the denotation is the same as the equivalence relation induced by the observable

- If a denotation is the best (most abstract) choice in the set of correct denotations

- A correct but non-minimal denotation might contain too many irrelevant details, which distinguish equivalent programs and might render it useless in more complex reasoning about the observable

- If a denotation is correct w.r.t. an observable \( \alpha_1 \), it is also correct to any "more abstract" observable \( \alpha_2 \)
• an important property of denotations, which allows one to reason on the properties of a program by reasoning on the properties of the (syntactic) program components.

• there exists an isomorphism between syntax and semantics

  $f : \text{syntactic operator}$

  $F : \text{semantic operator}$

  $[f(A_1, ..., A_m)] = F([A_1], ..., [A_m])$

• the typical definition style of denotational semantics

• compositionality can typically get lost when taking the least fixpoint.
A TRADITIONAL VIEW OF THE SYNTACTIC OPERATORS OF LOGIC LANGUAGES

goal

definite clauses

\[ \text{goal} \]

\[ \text{definite clauses} \]

\[ \text{program} = \text{set of definite clauses} = \text{set of procedure declarations} \]

\[ \text{AND} \]

\[ \text{the mechanism to syntactically compose procedure calls} \]

\[ \text{goal} = \text{a composition of procedure calls} \]

\[ \text{OR} \]

\[ \text{the mechanism to syntactically compose procedure declarations} \]

\[ \text{a program is a conjunction of clauses} \]

\[ \text{the operator \textsc{or} is called \textsc{or}, because it is used to represent a disjunction in the body of a rule clause defining a procedure} \]

\[ A = B, C \quad A \leftarrow (B \land C) \lor (D \land E) \]

\[ A = D, E \]

\[ \text{which compositional properties} \]

\[ \text{procedural compositionality} \]

\[ \text{from the denotation of a procedure to the denotation of a procedure call} \]

\[ \text{AND - compositionality} \]

\[ \text{from the denotation of a set of procedure calls to the denotation of their AND-composition (goal or clause body)} \]

\[ \text{OR - compositionality} \]

\[ \text{from the denotations of two sets of clauses to the denotation of the OR-composition of the two sets} \]
PROPERTIES OF THE LEAST HERBRAND MODEL

- $P$ is both correct and minimal w.r.t. the observable "success"

$\forall P_1, P_2 \quad H_{P_1} = H_{P_2} \iff W^a_{P_1} = W^a_{P_2}$

- Two programs have the same least Herbrand model if and only if they have the same set of irreducible goals

- Procedural compositionality holds
  - A procedure call succeeds if it has an instance in the denotation of the procedure (i.e., $w \in H_p$)

- AND-compositionality does not hold
  - The true behaviour of the goal $?-A \land B$ cannot be predicted from the true behaviours of $A$ and $B$

- Let us try with another observable

  "ground instances" of computed answer substitutions

  $d_{\text{mle}}^2 = \left\{ \langle G, v^d \rangle \mid
  \begin{array}{l}
  G \vdash ^d v^d \text{ is ground} \\
  G \vdash ^d \Box, \\
  G \vdash ^d \Box \text{ is an instance of } G \vdash ^d
  \end{array}\right\$
• Mp is both correct and minimal w.r.t. known instances of computed answers (d₂)

- **Minimal procedural compositionality holds**

\[ \mathcal{A} \models_{mp} \left\{ <A, x> \mid \exists B \in MP, \forall i \in \text{mp}_n(A_i, x) \text{ witnessed by } A \right\} \]

- **AND-compositionality holds**

\[ \mathcal{A}, \mathcal{G} \models_{mp} \text{ can be derived from } \mathcal{A} \models_{mp} \text{ and } \mathcal{G} \models_{mp} \]

- Procedural and AND-compositionality can be combined in a single theorem, which tells us that the observation for any goal G can be predicted by "executing" the goal in the oclamnation Mp

• **Goal-compositionality (condensing) theorem**

\[ \forall G = \{ G_1, \ldots, G_m \} \text{ goal} \]

\[ <G_1, \nu> \in MP \Rightarrow \nu \uparrow \in MP \]

\[ \exists \left\{ B_1, \ldots, B_n \right\} \subseteq MP \]

\[ \nu = \text{mp}_n \left( (A_1, \ldots, A_m), (B_1, \ldots, B_n) \right) \]

\[ \nu \uparrow \text{ is a formed instance of } G_1 \]

• d₁ (muenv) and d₂ (ground computed answers) define exactly the same equivalence relation

- Mp can be better be considered a oclamation for d₂, because of the compositionality properties

- OR-compositionality does not hold neither for d₁ nor for d₂

- It is necessary only when modular reasoning is required
Another proof-theoretic construction of $MP$

- The proof-theoretic (operational) characterization of $MP$ was the union set
  \[ \mathcal{O}_p = \{ A \land Bp \mid \neg A \text{ has an } \text{sub-reduction in } \text{MP} \} \]

- There exists another (equivalent) construction given in terms of the observable $\mathcal{O}_2$
  - collecting all the observations for most general atomic goals
  - procedure call, with no constraints on the inputs
  - and applying the computed observations to the initial goal

  \[ \mathcal{O}_p = \{ A \in Bp \mid A \text{ is a proper instance of } p(x_1, \ldots, x_n) \} \]

- This property holds for a large class of observables and will allow us to derive the proof-theoretic characterization of the denotation from the observable.
LEAST HERBRAND MODEL AND COMPUTED ANSWERS

- The observable computed answers
  \[ x_3 = \{ \langle G, t \rangle \mid G \vdash \frac{x_2}{2} \} \]

- \( M_p \) is not correct w.r.t. \( d_3 \)
  \[ M_{d_3} = M_{p_2} \neq W_{p_2}^{d_3} = W_{p_2} \]

Counterexample

- \( P_4 \)
  \[
  \begin{cases}
  p(a) \\
  q(x)
  \end{cases}
  \]

- \( P_2 \)
  \[
  \begin{cases}
  p(a) \\
  q(x) \\
  q(y)
  \end{cases}
  \]

\[ M_{d_3} = M_{p_2} = \{ p(a), q(a) \} \]

\[ W_{p_2}^{d_3} \neq W_{p_2}^{d_3}, \text{ because } \text{the pool has different answers in } P_4 \text{ and } P_2 \]

- \( \times \) in \( P_4 \)
- \( \times \) and \( \{ x \neq a \} \) in \( P_2 \)

- The problem might be related to the fact that \( M_p \), being a subset of the Herbrand Base, does not properly determine the behaviour of non-ground rules.

- A different denotation, defined on non-ground interpretations.
$M_p = O_p = \{ A \mid A \text{ is an instance of } p(x_1, \ldots, x_n) \}
= \{ A \mid A \text{ is a ground instance of } p(x_1, \ldots, x_n) \}
\vdash ?- p(x_1, \ldots, x_n) \frac{p(c)}{\square} \}$

obtainable "ground instances of computed answers"

$O_p^c = \{ A \mid A \text{ is an instance of } p(x_1, \ldots, x_n) \}
\vdash ?- p(x_1, \ldots, x_n) \frac{p(c)}{\square} \}$

obtainable "instances of computed answers = correct answers"

1. $O_p^c$ is a declarative interpretation
   - a ground atom in $O_p^c$ stands for its equivalent class w.r.t. evidence

2. The declarative view of $O_p^c$
   - the set of atomic logical consequences

3. $O_p^c$ is correct w.r.t. "correct answers" but is not correct w.r.t. computed answers

$P_1 \quad \begin{array}{c} p(c) \quad q(x) \end{array} 
\quad P_2 \quad \begin{array}{c} p(c) \\ p(x) \\ q(c) \end{array}$

$O_{P_1}^c = O_{P_2}^c = \{ p(c), q(x), q(a) \}$

4. In order to get a correct evaluation w.r.t. computed answers, we have to get not of instances in the definition of the proof-theoretic orientation
as in the case of \( \mathcal{M}_p \), we can give an equivalent definition of \( \mathcal{O}_p \) as least fixpoint of an immediate consequences operator:

\[
\mathcal{O}_p = \{ A \mid A \text{ is a ground instance of } p(x_1, \ldots, x_n) \}
\]

\[
\neg p(x_1, \ldots, x_n) \vdash \Box^p \top
\]

\[
T^p(I) = \{ A \mid A := B_1, \ldots, B_n \text{ is a ground instance of a clause in } \mathcal{P} \}
\]

\[
\mathcal{O}^p = \{ A \mid A = p(x_1, \ldots, x_n) \}
\]

\[
\neg p(x_1, \ldots, x_n) \vdash \Box^p \top
\]

\[
T^s(I) = \{ A \mid A := B_1, \ldots, B_n \text{ is a clause in } \mathcal{P} \}
\]

\[
\neg B_1, \ldots, B_n \vdash \Box^p \top
\]

\[
\mathcal{O}^p = T^s \uparrow \mathcal{U}
\]

\[
T^s \text{ is continuous}
\]
\( O_p = \{ A \mid A \text{ is a ground instance of } p(x_1, \ldots, x_n) \} \)
\( O_p^c = \{ A \mid A \text{ is an instance of } p(x_1, \ldots, x_n) \} \)
\( O_p^s = \{ A \mid A = p(x_1, \ldots, x_n) \} \)

- Some semantic domain of the C-semantics
- \( O_p^s \) is correctly w.r.t. computed answers
- Procedural compositionality holds
- AND-compositionality holds
- Pool-compositionality (concluding) theorem

\( \forall G = \ldots, A_n \)
\( \langle G, \forall \rangle \in Wp^d \)
\( G \rightarrow \psi \)

if and only if

\( \exists \ (B_1, \ldots, B_n) \in O_p^s \)
\( (A_1 = \text{magn}((A_2, \ldots, A_n), (B_2, \ldots, B_n)), G^+ = G^{+1}) \)

- Once we have computed the answers for most general atomic pools \( O_p^s \), the answers for any pool \( G \) can be obtained, by "executing" the pool in the S-semantics.
Properties of the S-Semantics

1. The declarative concept of correct answers (given in terms of all the models) has a characterization in terms of one model only (S-semantics or S-semantics).
   - This is not true for Mp.

2. The S-semantics is language independent.
   - If we add new constant and function symbols, the S-semantics is not affected, while the least Herbrand model and the communities are.

3. It is not true that Mp does not correctly models computed answers only for artificial maintaining programs.
   - $Mp = Op^3$ holds if and only if $P$ is language independent.

4. A decidable approximation is allowed programs.
   - Ground unit clauses
   - No hypotheses partially evaluated
   - Partially evaluated
   - No deductive rules into

5. Any way, even the S-semantics is not always the best choice.
   - The S-semantics is not OR-conventional.
Towards an or-compositional semantics for computed answers

- $O^s_p$ is not or-compositional

- The s-semantics of $P_1 \lor P_2$ cannot be derived from the s-semantics of $P_1$ and $P_2$

\[
P_1 \quad p(x) \leftarrow r(x) \\
p(a)
\]

\[
P_2 \quad \{r(b)\}
\]

\[
O^s_{P_1} = \{p(a)\} \\
O^s_{P_2} = \{r(b)\} \\
O^s_{P_1 \lor P_2} = \{p(a), p(b), r(b)\}
\]

- $T^s_p$ is or-compositional (by construction), but $T^s_p \uparrow$ is not

- Or-compositionality requires to maintain the relations among predicates

  - $T^s_p$ is a function from interpretation to interpreting ($T^s_p!$)
  - by means of clauses

- Or-compositionality can be embedded into the definition of observational equivalence (${\leq}_{op} = \text{compositional computable answer}$)

$P_1 \equiv P_2$

∀G goal, ∀P program

\[G \xrightarrow{P_1 \lor P_2} \square \quad G \xrightarrow{P_2 \lor P} \square\]

G is increment of G by
\( O_p^{d_4} = \{ \begin{align*} & p(x_1, \ldots, x_n) \Downarrow \vdash B_2, \ldots, B_n \\ & ? p(x_1, \ldots, x_n) \xrightarrow{\varphi} ? B_2, \ldots, B_n \end{align*} \} \)

- Correctness

\[
\text{if } O_{p_2}^{d_4} = O_{p_2}^{d_4} \implies P_2 \equiv_{d_4} P_2
\]

- Or-compositionality

\[
\forall \varnothing, \quad O_{p_2 \cup p_2} = O_{p_2 \cup p_2}
\]

- Is not minimal (and therefore not fully abstract)

\[
T_{p}^{d_4}(I) = \left\{ \begin{array}{l}
\frac{1}{2} < 1 \\
A: = B_2, \ldots, B_n \in P \\
H_2, \ldots, H_5, H_5: = B_2, \ldots, Gx \in I \\
\varphi = mgn((B_2, \ldots, B_5), (H_2, \ldots, H_5)) \\
C = (A: = Gx, \ldots, Gx, B_5x, \ldots, B_n) \quad \varnothing \end{array} \right\}
\]

\[
T_{p}^{d_4} \uparrow W = O_{p}^{d_4}
\]
The same semantics which characterize computed answers in an OK-conditional way so model a different observable, i.e. constraints.

Any intermediate state of an SLD-resolution can be represented by a formula

\[ B_1 \land \ldots \land B_N \Rightarrow (A_1, \ldots, A_n) \leq \]

A restriction to the initial goal of the composition of the inputs.

If the initial goal is atomic, a resultant is a Horn clause

\[ A_1 \land \ldots \land A_n \Rightarrow B_1, \ldots, B_N \]

The reduction step.
$O_{p^4} = T_{d^4} \uparrow w$ is
- correct
- minimal
- DR-compositionally wrt. $d_{5}$ (comments)

* the goal-compositionality (conditioning) lemma

$c$ is a kernel of $? - A_2, \ldots, A_m$ in $P$ iff

$$\exists \{ H_2, \ldots, H_{s-4}, H_s : B_2, \ldots, B_m \} \in O_{p^4}$$

$$\forall = mru \left( (A_2, A_5), (H_2, \ldots, H_s) \right)$$

$$c' = (A_2 \land \ldots \land A_m) \land B_2, \ldots, B_m, (A_{s+2}, \ldots, A_m) \forall$$

$c'$ is a variant of $c$

**OTHER (MORE ABSTRACT) OBSERVABLES**

- cell patterns
  - procedure calls

- partial answers
  - answers computed at intermediate steps

  * some blowup of the &-semantics
1. Top-down definition
   
   Collect all the observables for the goals of the form \( p(x_1, \ldots, x_n) \)

2. (Equivalent) bottom-up definition

3. The denotations are goal independent

4. The observable for a specific goal \( G \) can be determined by "executing" \( G \) in the denotation

5. The construction can be useful for abstract interpretation
from the purely logical viewpoint we don't care about issues like observational equivalence and compositionality.

The standard denotation is one

if the denotation is to be used in

semantics - being program manipulation

we are forced to be concerned with the more

computational issues

- Program transformations should preserve at least computed answers, and hence a denotation correct wrt computed answers.

- a program analysis aiming at establishing properties of the computed answers (e.g. aliasing or randomness analysis) should abstract a denotation correct wrt computed answers.

- a modular program analysis technique requires an or-compositional denotation.

- a program analysis aiming at establishing properties of the procedure calls (for optimization purposes) needs a denotation correct wrt observable which shows more internal computation details.

- many different semantics containing more information than the purely logical information of the standard denotation.