

Logic Programming languages

Logic Programming Languages

Preliminaries

Syntax

- Syntax: *extended Backus-Naur form* (EBNF) for notation of production rules

Name: $G, H ::= A \mid B, \text{Condition}$

- ▶ *capital letters*: syntactical entities
- ▶ *symbols G and H defined*: with name *Name*.
- ▶ *Condition holds*: G and H can be of the form A or B

Semantics

Operational (Procedural) Semantics

State transition system (refined calculus)

Declarative Semantics

Logical reading of program as theory, i.e. set of formulae of first-order logic

Operational vs. Declarative Semantics

- Soundness
- Completeness

Simple State Transition System

- (S, \mapsto)
 - ▶ S set of states
 - ▶ \mapsto binary relation over states: *transition relation*
 - ▶ (*state*) *transition* from S_1 to S_2 possible if (some given) condition holds, i.e. S_1 and S_2 are in relation \mapsto , written $S_1 \mapsto S_2$
- distinguished subsets of S : *initial* and *final states*
- $S_1 \mapsto S_2 \mapsto \dots \mapsto S_n$ *derivation (computation)*
- \mapsto^* reflexive-transitive closure of \mapsto

Reduction is synonym for transition

Transition Rules

- The relation \mapsto is defined by using transition rules of the form

IF	<i>Condition</i>
THEN	$S \mapsto S'$.

- (*state*) *transition (reduction, derivation step, computation)* from S to S' possible if the condition *Condition* holds
- straightforward to concretize a calculus (" \vdash ") into a state transition system (" \mapsto ")

Operational (Procedural) Semantics

Refined Calculus

Triple (Σ, \equiv, T)

- Σ : signature for a first-order logic language
- \equiv : *congruence* (equivalence relation) on states
- $T = (S, \mapsto)$: simple state transition system
states S represent logical expressions over the signature Σ

Congruence

- states which are considered equivalent for purpose of computation
- *congruence* instead of modeling with additional transition rules
- formally congruence is equivalence relation compatible with the structure:

(Reflexivity) $A \equiv A$

(Symmetry) If $A \equiv B$ then $B \equiv A$

(Transitivity) If $A \equiv B$ and $B \equiv C$ then $A \equiv C$

Example:

$$(X = 3) \equiv (3 = X)$$

Congruence

<i>Commutativity:</i>	$G_1 \wedge G_2$	\equiv	$G_2 \wedge G_1$
<i>Associativity:</i>	$G_1 \wedge (G_2 \wedge G_3)$	\equiv	$(G_1 \wedge G_2) \wedge G_3$
<i>Identity:</i>	$G \wedge \top$	\equiv	G
<i>Absorption:</i>	$G \wedge \perp$	\equiv	\perp

derived from tautologies

Transition Rules for Resolution Calculus

(1)

- *state*: set of clauses
- *initial state*: clauses representing the theory and the negated consequence
- *final state*: contains the empty clause

Transition Rules for Resolution Calculus.

Propositional Logic

- **Resolvent**

IF $R \vee A \in S$ and $R' \vee \neg A \in S$ for some atom A
THEN $S \mapsto S \cup \{F\}$ where $F = R \vee R'$ is called Resolvent.

- **Factor**

IF $R \vee L \vee L \in S$ for some literal L
THEN $S \mapsto S \cup \{F\}$ where $F = R \vee L$ is called Factor

Transition Rules for Resolution Calculus.

FOL

- **Resolvent**

IF $R \vee A \in S$ and $R' \vee \neg A \in S$ and
 σ is a most general unifier for the atoms A and A' ,
THEN $S \mapsto S \cup \{F\}$ where $F = (R \vee R')\sigma$ is called Resolvent.

- **Factor**

IF $R \vee L \vee L' \in S$ and
 σ is a most general unifier for the literals L and L' ,
THEN $S \mapsto S \cup \{F\}$ where $F = (R \vee L)\sigma$ is called
Factor

Logic Programming

A logic program is a set of axioms, or rules, defining relationships between objects. A computation of a logic program is a deduction of consequences of the program. A program defines a set of consequences, which is its meaning. The art of logic programming is constructing concise and elegant programs that have desired meaning.

Sterling and Shapiro: The Art of Prolog, Page 1.

- *goal*:
 - ▶ *empty goal* \top (top) or \perp (bottom), or
 - ▶ atom, or
 - ▶ conjunction of goals
- (*Horn*) *clause*: $A \leftarrow G$
 - ▶ *head* A : atom
 - ▶ *body* G : goal
- Naming conventions
 - ▶ *fact*: clause of form $A \leftarrow \top$
 - ▶ *rule*: all others
- (*logic*) *program*: finite set of Horn clauses
- predicate symbol *defined*: it occurs in head of a clause

LP Calculus – Syntax

<i>Atom:</i>	A, B	$::=$	$p(t_1, \dots, t_n), n \geq 0$
<i>Goal:</i>	G, H	$::=$	$\top \mid \perp \mid A \mid G \wedge H$
<i>Clause:</i>	K	$::=$	$A \leftarrow G$
<i>Program:</i>	P	$::=$	$K_1 \dots K_m, m \geq 0$

LP Calculus – State Transition System

- *state* $\langle G, \theta \rangle$
 - ▶ G : goal
 - ▶ θ : substitution
- *initial state* $\langle G, \epsilon \rangle$
- *successful final state* $\langle \top, \theta \rangle$
- *failed final state* $\langle \perp, \epsilon \rangle$

Derivation is

- *successful*: its final state is successful
- *failed*: its final state is failed
- *infinite*: if there are an infinite sequence of states and transitions $S_1 \mapsto S_2 \mapsto S_3 \mapsto \dots$

Goal G is

- *successful*: it has a successful derivation starting with $\langle G, \epsilon \rangle$
- *finitely failed*: has only failed derivations starting with $\langle G, \epsilon \rangle$

Logical Reading, Answer

- If $\langle G, \epsilon \rangle \mapsto^* \langle H, \theta \rangle$ then the *logical reading* of $\langle H, \theta \rangle$ is
 - ▶ $\exists \bar{X}(H\theta)$
 - ▶ where \bar{X} are the variables which occur in $H\theta$ but not in G
- *Computed answer substitution (cas)* of a goal G is defined as: a substitution θ such that there exists with successful derivation $\langle G, \epsilon \rangle \mapsto^* \langle \top, \theta \rangle$

G is also called *initial goal* or *query*

Operational Semantics

Unfold

If $(B \leftarrow H)$ is a fresh variant of a clause in P
and β is the most general unifier of B and $A\theta$
then $\langle A \wedge G, \theta \rangle \mapsto \langle H \wedge G, \theta\beta \rangle$

Failure

If there is no clause $(B \leftarrow H)$ in P
with a unifier of B and $A\theta$
then $\langle A \wedge G, \theta \rangle \mapsto \langle \perp, \epsilon \rangle$

Non-determinism

The **Unfold** transition exhibits two kinds of non-determinism.

- *don't-care non-determinism*:
 - ▶ any atom in $A \wedge G$ can be chosen as the atom A according to the congruence defined on states
 - ▶ affects length of derivation (infinitely in the worst case)
- *don't-know non-determinism*:
 - ▶ any clause $(B \leftarrow H)$ in P for which B and $A\theta$ are unifiable can be chosen
 - ▶ determines the computed answer of derivation

Unfold with Case Splitting

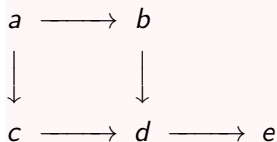
UnfoldSplit

If $(B_1 \leftarrow H_1), \dots, (B_n \leftarrow H_n)$ are fresh variants
of all those clauses in P for which
 B_i ($1 \leq i \leq n$) is unifiable with $A\theta$
and β_i is the most general unifier of B_i and $A\theta$ ($1 \leq i \leq n$)
then $\langle A \wedge G, \theta \mapsto \rangle \langle H_1 \wedge G, \theta\beta_1 \rangle \quad \dots \quad \langle H_n \wedge G, \theta\beta_n \rangle$

SLD Resolution

- *selection strategy*: textual order of clauses and atoms in a program
- *(chronological) backtracking (backtrack search)*
- left-to-right, depth-first exploration of the search tree
- efficient implementation using a stack-based approach
- can get trapped in infinite derivations
(but breadth-first search far too inefficient)

Example - Accessibility in DAG



$\text{edge}(a,b) \leftarrow \top \text{ (e1)}$

$\text{edge}(a,c) \leftarrow \top \text{ (e2)}$

$\text{edge}(b,d) \leftarrow \top \text{ (e3)}$

$\text{edge}(c,d) \leftarrow \top \text{ (e4)}$

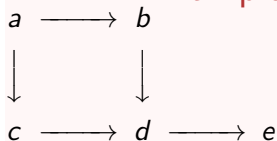
$\text{edge}(d,e) \leftarrow \top \text{ (e5)}$

$\text{path}(\text{Start},\text{End}) \leftarrow \text{edge}(\text{Start},\text{End}) \text{ (p1)}$

$\text{path}(\text{Start},\text{End}) \leftarrow \text{edge}(\text{Start},\text{Node}) \wedge \text{path}(\text{Node},\text{End}) \text{ (p2)}$

Note: e1 en and p1, p2 are names of rules in the metalanguage, they are part of the LP syntax.

Example - Accessibility in DAG (cont)

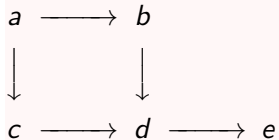


$\langle \text{path}(b, Y), \varepsilon \rangle$
 $\mapsto (p1) \langle \text{edge}(S, E), \{S \mapsto b, E \mapsto Y\} \rangle$
 $\mapsto (e3) \langle \top, \{S \mapsto b, E \mapsto d, Y \mapsto d\} \rangle$

With the second rule $p2$ for path selected:

$\langle \text{path}(b, Y), \varepsilon \rangle$
 $\mapsto (p2) \langle \text{edge}(S, N) \wedge \text{path}(N, E), \{S \mapsto b, E \mapsto Y\} \rangle$
 $\mapsto (e3) \langle \text{path}(N, E), \{S \mapsto b, E \mapsto Y, N \mapsto d\} \rangle$
 $\mapsto (p1) \langle \text{edge}(N, E), \{S \mapsto b, E \mapsto Y, N \mapsto d\} \rangle$
 $\mapsto (e5) \langle \top, \{S \mapsto b, E \mapsto e, N \mapsto d, Y \mapsto e\} \rangle$

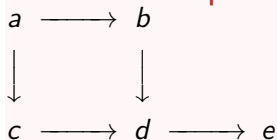
Example - Accessibility in DAG (cont 2)



Partial search tree:

As exercise

Example - Accessibility in DAG (cont 2)



With the first rule:

$$\begin{aligned} & \langle \text{path}(f, g), \varepsilon \rangle \\ \mapsto_{(p1)} & \langle \text{edge}(S, E), \{S \mapsto f, E \mapsto g\} \rangle \\ \mapsto & \langle \perp, \varepsilon \rangle \end{aligned}$$

With the second rule (and special selection) we get an infinite derivation:

$$\begin{aligned} & \langle \text{path}(f, g), \varepsilon \rangle \\ \mapsto_{(p2)} & \langle \text{path}(N, E) \wedge \text{edge}(S, N), \{S \mapsto f, E \mapsto g\} \rangle \\ \mapsto_{(p2)} & \langle \text{edge}(S, N) \wedge \text{edge}(N, N1) \wedge \text{path}(N1, E), \{S \mapsto f, E \mapsto g\} \rangle \end{aligned}$$

Declarative Semantics

- *implication* ($G \rightarrow A$): Horn clause, definite clause
- *logical reading of a program* P : universal closure of the conjunction of the clauses of P , written P^{\rightarrow}
- only positive information can be derived
- “complete” P^{\rightarrow} :
 - ▶ keep *necessary* conditions (implications)
 - ▶ add corresponding *sufficient* conditions (implications in the other direction)

Example:

In DAG with nodes a, b, c, d, e : $P^{\rightarrow} \not\models$

$\text{path}(f, g) = \text{true}, P^{\rightarrow} \not\models \neg \text{path}(f, g) = \text{true}$ In completed logical reading, $P^{\rightarrow} \models$

Completion of P : P^{\leftrightarrow}

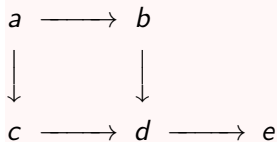
For each p/n in P add to P^{\leftrightarrow} the formula

$$\begin{array}{llll} p(\bar{t}_1) \leftarrow G_1 & \forall \bar{X} (p(\bar{X}) \leftrightarrow \exists \bar{Y}_1 (\bar{t}_1 \doteq \bar{X} \wedge G_1)) & \vee \\ \vdots & \vdots & \vee \\ p(\bar{t}_m) \leftarrow G_m, & \exists \bar{Y}_m (\bar{t}_m \doteq \bar{X} \wedge G_m)), & \vee \end{array}$$

- \bar{X} : pairwise distinct fresh variables
- \bar{t}_i : terms
- \bar{Y}_i : variables occurring in G_i and t_i

Add $\forall \bar{X} \neg p(\bar{X})$ to P^{\leftrightarrow} for all undefined predicate symbols p in P .

Example – Logical Reading



For the edge/2 predicates

$\text{edge}(a,b) \leftarrow \top$

\vdots

$\text{edge}(d,e) \leftarrow \top$

add to P^{\leftrightarrow} the formula

$\forall X_1 X_2 \quad (\text{edge}(X_1, X_2) \leftrightarrow$
 $(a \doteq X_1, b \doteq X_2) \quad \vee$

$\vdots \quad \vee$
 $(d \doteq X_1, e \doteq X_2))$

Example – Logical Reading (2)

For the path/2 predicates

add to P^{\leftrightarrow} the formula

$\text{path}(\text{Start}, \text{End}) \leftarrow$
 $\text{edge}(\text{Start}, \text{End})$

$\text{path}(\text{Start}, \text{End}) \leftarrow$
 $\text{edge}(\text{Start}, \text{Node}) \wedge$
 $\text{path}(\text{Node}, \text{End})$

$$\begin{aligned} & \forall X_1 X_2 (\text{path}(X_1, X_2) \leftrightarrow \\ & \exists Y_{11} Y_{12} (Y_{11} \doteq X_1, Y_{12} \doteq X_2 \\ & \wedge \text{edge}(Y_{11}, Y_{12})) \vee \\ & \exists Y_{21} Y_{22} Y_{23} (Y_{21} \doteq X_1, Y_{22} \doteq X_2 \\ & \wedge \text{edge}(Y_{21}, Y_{23}) \\ & \wedge \text{path}(Y_{23}, Y_{22})) \\ & \leftrightarrow \\ & \text{edge}(X_1, X_2) \vee \\ & (\exists Y \text{ edge}(X_1, Y) \vee \text{path}(Y, X_2))) \end{aligned}$$

Clark's Equality Theory (CET)

Universal closure of the formulae

Reflexivity $(\top \rightarrow X \doteq X)$

Symmetry $(X \doteq Y \rightarrow Y \doteq X)$

Transitivity $(X \doteq Y \wedge Y \doteq Z \rightarrow X \doteq Z)$

Compatibility $(X_1 \doteq Y_1 \wedge \dots \wedge X_n \doteq Y_n \rightarrow f(X_1, \dots, X_n) \doteq f(Y_1, \dots, Y_n))$

Decomposition $(f(X_1, \dots, X_n) \doteq f(Y_1, \dots, Y_n) \rightarrow X_1 \doteq Y_1 \wedge \dots \wedge X_n \doteq Y_n)$

Contradiction (Clash) $(f(X_1, \dots, X_n) \doteq g(Y_1, \dots, Y_m) \rightarrow \perp) \quad \text{if } f \neq g$

Acyclicity $(X \doteq t \rightarrow \perp) \quad \text{if } t \text{ is function term and } X \text{ appears in } t$

(Σ signature with infinitely many functions, including at least one constant)

Unifiable in CET

Terms s and t are unifiable if and only if

$$CET \models \exists (t \dot{=} s).$$

Examples:

$X \doteq X$ is unifiable but *not*:

- $X \doteq f(X)$
- $X \doteq p(A, f(X, a))$
- $X \doteq Y \wedge X \doteq f(Y)$

Soundness and Completeness

- **Soundness:**

If θ is a computed answer of G , then $P^{\leftrightarrow} \cup CET \models \forall G\theta$.

- **Completeness:**

If $P^{\leftrightarrow} \cup CET \models \forall G\theta$, then a computed answer σ of G exists, such that $\theta = \sigma\beta$.

(P logic program, G goal, θ substitution)

Failed Derivations

- **Fair Derivation:**

Either fails or each atom appearing in the derivation is selected after finitely many reductions.

- **Soundness and Completeness:**

Any fair derivation starting with $\langle G, \epsilon \rangle$ fails finitely if and only if

$$P^{\leftrightarrow} \cup CET \models \neg \exists G.$$

Remarks:

- not valid without CET
- SLD resolution not fair

(P logic program, G goal)