

# Version Space Algebra in Program Synthesis

Shaoyuan Chen

SPAR PL4SE, Institute of Computer Software, Nanjing University

March 12, 2020

# Version Space

In machine learning, the **version space method** proposed by Mitchell [4] learns a Boolean function from given positive/negative examples. Lau et. al. extended Mitchell's version space concept to functions with arbitrary range and defined **version space algebra** [3]. Using these techniques, they developed SMARTedit, a *programming by demonstration* application for repetitive text editing tasks.

Version space algebra is especially useful in

- 1 programming by demonstration (PbD);
- 2 synthesizing action sequences (e.g., text-editing scripts [4], robot control programs [5]).

It can even be used to synthesize shell scripts [6] and simple python programs [1].

# Terminology

**Hypothesis** a function (from attribute space to label space) in machine learning; a program (from input space to output space) in program synthesis.

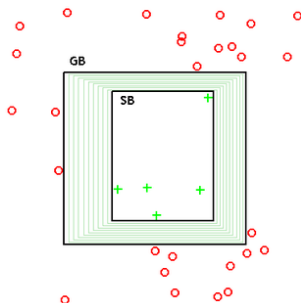
**Hypothesis space** the set of all functions can be learned by a learning algorithm; the set of all programs can be produced by a synthesizer (= search space).

**Version Space** the set of all hypotheses in a hypothesis space  $H$  consistent with all training examples in a training set  $D$ , referred as  $VS_{H,D}$ .

## Example: Binary Classification with Rectangle

Problem: given a set of points with binary labels in a plane, find a rectangle that separates all positive points and all negative sample points.

- Hypothesis: all functions  $h : \mathbb{R}^2 \rightarrow \{0, 1\}$
- Hypothesis space:  $\{h \mid \{(x, y) : h(x, y) = 1\} = [A, B] \times [C, D]\}$



# Candidate Elimination Algorithm

The *candidate elimination algorithm* is used to compute the version space  $VS_{H,D}$  given hypothesis space  $H$  and training set  $D$ .

---

**Algorithm 1** Candidate elimination algorithm

---

**Input:** hypothesis space  $H$ ; training set  $D$

**Output:** version space  $VS_{H,D}$

- 1:  $VS \leftarrow H$
  - 2: **for** each training sample  $(x, y)$  in  $D$  **do**
  - 3:      $VS \leftarrow \{h \in VS \mid h(x) = y\}$  ▷ Refinement step
  - 4: **end for**
  - 5: **return**  $VS$
- 

Maintaining the version space as a list of hypotheses is often infeasible, because the size of the hypothesis space may be very large. We need to find a *compressed* representation of a version space to speed up the refinement step.

# Hypotheses Space as a Poset

We may define a partial order between the hypotheses in a hypothesis space.

For Boolean-valued functions, a canonical partial order has been defined.

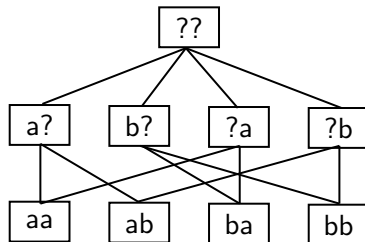
## Definition (generality order)

For hypotheses with Boolean values, we say  $h_1$  is **more general** than  $h_2$  ( $h_1 \succeq h_2$ ) if  $h_2(x) \rightarrow h_1(x)$  for all  $x$  in input space. The induced partial order in a hypothesis space is called the **generality order**.

For arbitrary-valued functions, the partial order may be defined by the application designer.

## Example: the “ab?” Language

- Hypothesis:  $\{a, b\}^2 \rightarrow \{0, 1\}$
- Hypothesis space:  $\{a, b, ?\}^2$ , where ? matches any character
- Hasse diagram of the generality order:



# Boundary-Set Representability

With a partial order defined on the hypothesis space, we may use two antichains  $G, S$  (called *boundaries*) to represent a version space.

## Definition (boundary-set representability)

A version space  $VS$  of a partially-ordered hypothesis space  $H$  is **boundary-set representable (BSR)**, if it can be written as the form

$$VS = \{h \in H \mid \exists h_g \in G, \exists h_s \in S, h_g \succeq h \succeq h_s\}$$

where  $G, S$  are two antichains of  $H$ .

Examples for the “ab?” language:

- $D = \{(ab, 1)\}$ :  $G = \{??\}$ ,  $S = \{ab\}$ ;
- $D = \{(aa, 0)\}$ :  $G = \{b?, ?a\}$ ,  $S = \{ab, ba, bb\}$ ;
- $D = \{(aa, 1), (ab, 1)\}$ :  $G = S = \{a?\}$ .



# Boundary-Set Representability

Not all version spaces are BSR. Hirsh showed that **convexity** and **definiteness** are a necessary and sufficient condition for a version space being BSR [2].

A subset  $C$  of a partially ordered set  $(S, \preceq)$  is

- **convex**, if:  $x \preceq y \preceq z$  ( $x, z \in C, y \in S$ ) implies  $y \in C$ ;
- **definite**, if: for every  $y \in C$ , there exists  $x \in \min\{C\}$  (minimal elements of  $C$ ) and  $z \in \max\{C\}$  (maximal elements of  $C$ ), such that  $x \preceq y \preceq z$ .

## Theorem (version space representation theorem)

*Every version space of a Boolean hypothesis space with generality order is BSR.*

# Boundary-Set Representability

With boundary-set representation, the version space can be updated in a more efficient approach.

Example:

- Hypothesis:  $\Sigma^* \rightarrow \{0, 1\}$  where  $\Sigma = \{a, b, c\}$
  - Hypothesis space:  $\{s^* \mid s \in \Sigma^*\}$
- 
- ①  $D = \{(aba, 1)\}$ ,  $G = \{*\}$ ,  $S = \{aba^*\}$ ;
  - ②  $D = \{(aba, 1), (ba, 0)\}$ ,  $G = \{a^*\}$ ,  $S = \{aba^*\}$ ;
  - ③  $D = \{(aba, 1), (ba, 0), (abc, 1)\}$ ,  $G = \{a^*\}$ ,  $S = \{ab^*\}$ ;

# Version Space Algebra

Lau et. al. defined **version space algebra**, which allows us to build complex version space from simple, atomic version spaces.

## Definition (version space union)

Given two hypothesis spaces  $H_1, H_2$  with the same input and output spaces, the **union** of two version spaces  $VS_{H_1, D} \cup VS_{H_2, D}$  is defined as  $VS_{H_1 \cup H_2, D}$ .

## Definition (version space join)

Given two version spaces  $VS_{H_1, D_1}$  and  $VS_{H_2, D_2}$ , the **independent join** of the two spaces  $VS_{H_1, D_1} \bowtie VS_{H_2, D_2}$  is defined as

$$\{\langle h_1, h_2 \rangle \mid h_1 \in VS_{H_1, D_1}, h_2 \in VS_{H_2, D_2}\},$$

where  $\langle h_1, h_2 \rangle(x, y) \equiv (h_1(x), h_2(y))$ .

# Version Space Algebra

The version space algebra is analogous to the context-free grammar, in the following sense:

<b>VSA</b>	<b>CFG</b>
atomic version space	terminal symbol
compound version space	non-terminal symbol
version space union	“or” of two production rules
version space join	concatenation of two symbols

Note: compound version spaces can be represented by representing all its components simultaneously; thus compound version space can be updated by updating all components individually.

# Example: SMARTedit

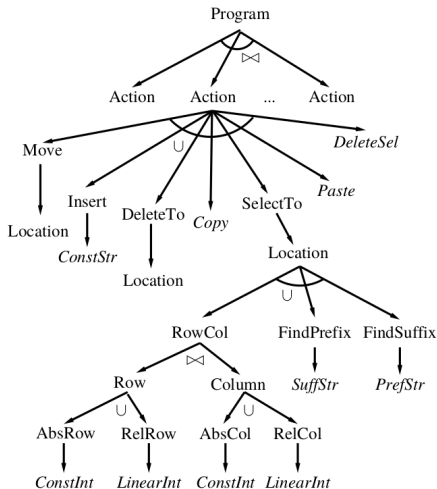


Figure: Version space structure for SMARTedit

# References

- [1] Pedro Domingos and Daniel Weld. Learning programs from traces using version space algebra. In *K-CAP*, 2003.
- [2] H. Hirsh. Theoretical underpinnings of version spaces. In *IJCAI*, 1991.
- [3] Tessa A. Lau, Pedro Domingos, and Daniel S. Weld. Version space algebra and its application to programming by demonstration. In *ICML*, 2000.
- [4] T. Mitchell. Generalization as search. *Artificial Intelligence*, 1982.
- [5] Michael Pardowitz, Bernhard Glaser, and Rüdiger Dillmann. Learning repetitive robot programs from demonstrations using version space algebra. In *International Conference on Robotics and Applications*, 2007.
- [6] Jyoti Yadav, Jyoti Chandel, and Neha Gupta. Programming shell scripts by demonstration. In *Workshop on Supervisory Control of Learning & Adaptive Systems, AAI*, 2004.