Learning Single Rules

- Introduction
  - Concept Learning
  - Generality Relations
  - Refinement Operators
  - Structured Hypothesis Spaces

- Simple algorithms
  - Find-S
  - Find-G

- Version Spaces
  - Version Spaces
  - Candidate-Elimination Algorithm

- Batch Learning
Concept

- Attribute-Value Representation
  - each object is represented with a finite number of attributes

- Concept
  - A concept is a subset of all possible objects

- Example 1:
  - objects are points in a 2-d plane
  - a concept can be any subarea in the plane
    - can have many disconnected components
  - # objects and # concepts is infinite

- Example 2:
  - all attributes are Boolean, objects are Boolean vectors
  - a concept can be any subset of the set of possible objects
  - # concepts and # objects is finite
Concept Learning

- Given:
  - Positive Examples $E^+$
    - examples for the concept to learn (e.g., days with golf)
  - Negative Examples $E^-$
    - counter-examples for the concept (e.g., days without golf)
  - Hypothesis Space $H$
    - a (possibly infinite) set of candidate hypotheses
    - e.g., rules, rule sets, decision trees, linear functions, neural networks, ...

- Find:
  - Find the target hypothesis $h \in H$
  - the target hypothesis is the concept that was used (or could have been used) to generate the training examples
Correctness

- What is a good rule?
  - Obviously, a correct rule would be good
  - Other criteria: interpretability, simplicity, efficiency, ...

- Problem:
  - We cannot compare the learned hypothesis to the target hypothesis because we don't know the target
    - Otherwise we wouldn't have to learn...

- Correctness on training examples
  - completeness: Each positive example should be covered by the target hypothesis
  - consistency: No negative example should be covered by the target hypothesis

- But what we want is correctness on all possible examples!
Conjunctive Rule

\[ \text{if } (\text{att}_i = \text{val}_{i,I}) \text{ and } (\text{att}_j = \text{val}_{j,J}) \text{ then } + \]

<table>
<thead>
<tr>
<th>Body of the rule (IF-part)</th>
<th>Head of the rule (THEN-part)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• contains a conjunction of conditions</td>
<td>• contains a prediction</td>
</tr>
<tr>
<td>• a condition typically consists of comparison of attribute values</td>
<td>• typically + if object belongs to concept, – otherwise</td>
</tr>
</tbody>
</table>

- Coverage
  - A rule is said to **cover** an example if the example satisfies the conditions of the rule.

- Prediction
  - If a rule covers an example, the rule's head is predicted for this example.
Propositional Logic

- simple logic of propositions
- combination of simple facts
- no variables, no functions, no relations
  \((\rightarrow\) predicate calculus)\
- Operators:
  - conjunction \(\wedge\), disjunction \(\vee\), negation \(\neg\), implication \(\rightarrow\), ...

- rules with attribute/value tests may be viewed as statements in propositional logic
  - because all statements in the rule implicitly refer to the same object
  - each attribute/value pair is one possible condition

- Example:
  - if \(\text{windy} = \text{false}\) and \(\text{outlook} = \text{sunny}\) then \(\text{golf}\)
  - in propositional logic: \(\neg\ \text{windy} \wedge \text{sunny}_{-}\text{outlook} \rightarrow \text{golf}\)
Generality Relation

- Intuitively:
  - A statement is more general than another statement if it refers to a superset of its objects
- Examples:

  All students are good in Machine Learning.
  All students who took a course in Machine Learning and Data Mining are good in Machine Learning
  All students who took course ML&DM at the TU Darmstadt are good in Machine Learning
  All students who took course ML&DM at the TU Darmstadt and passed with 2 or better are good in Machine Learning.
Generality Relation for Rules

- Rule \( r_1 \) is *more general* than \( r_2 \) \( r_1 \geq r_2 \)
  - if it covers all examples that are covered by \( r_2 \).
- Rule \( r_1 \) is *more specific* than \( r_2 \) \( r_1 \leq r_2 \)
  - if \( r_2 \) is more general than \( r_1 \).
- Rule \( r_1 \) is *equivalent* to \( r_2 \) \( r_1 \equiv r_2 \)
  - if it is more specific and more general than \( r_2 \).

Examples:
- if size > 5 then +
- if size > 3 then +
- if outlook = sunny then +
- if outlook = sunny and windy = false then +
- if animal = mammal then +
- if feeds_children = milk then +
Special Rules

- **Most general rule** $\top$
  - typically the rule that covers all examples
  - the rule with the body true
  - if disjunctions are allowed: the rule that allows all possible values for all attributes

- **Most specific rule** $\bot$
  - typically the rule that covers no examples
  - the rule with the body false
  - the conjunction of all possible values of each attribute
    - evaluates to false (only one value per attribute is possible)

- **Each training example can be interpreted as a rule**
  - body: all attribute-value tests that appear inside the example
  - the resulting rule is an immediate generalization of $\bot$
    - covers only a single example
Structured Hypothesis Space

The availability of a generality relation allows to structure the hypothesis space:

Structured Hypothesis Space

arrows to represent „is more general than“

Instance Space
Testing for Generality

- In general, we cannot check the generality of hypotheses
  - We do not have all examples of the domain available (and it would be too expensive to generate them)
- For single rules, we can approximate generality via a syntactic generality check:
  - Example: Rule \( r_1 \) is more general than \( r_2 \) if the set of conditions of \( r_1 \) forms a subset of the set of conditions of \( r_2 \).
  - Why is this only an approximation?
- For the general case, computable generality relations will rarely be available
  - E.g., rule sets
- Structured hypothesis spaces and version spaces are also a theoretical model for learning
Refinement Operators

- A refinement operator modifies a hypothesis
  - can be used to search for good hypotheses

- Generalization Operator:
  - Modify a hypothesis so that it becomes more general
    - e.g.: remove a condition from the body of a rule
  - necessary when a positive example is uncovered

- Specialization Operator:
  - Modify a hypothesis so that it becomes more specific
    - e.g., add a condition to the body of a rule
  - necessary when a negative examples is covered

- Other Refinement Operators:
  - in some cases, the hypothesis is modified in a way that neither generalizes nor specializes
    - e.g., stochastic or genetic search
Generalization Operators for Symbolic Attributes

There are different ways to generalize a rule, e.g.:

- **Subset Generalization**
  - a condition is removed
  - used by most rule learning algorithms

- **Disjunctive Generalization**
  - another option is added to the test

- **Hierarchical Generalization**
  - a generalization hierarchy is exploited

\[
\begin{align*}
\text{shape} &= \text{square} \land \text{color} = \text{blue} \rightarrow + \\
&\quad \Rightarrow \\
\text{color} &= \text{blue} \rightarrow +
\end{align*}
\]

\[
\begin{align*}
\text{shape} &= \text{square} \land \text{color} = \text{blue} \rightarrow + \\
&\quad \Rightarrow \\
\text{shape} &= (\text{square} \lor \text{rectangle}) \\
&\quad \land \text{color} = \text{blue} \rightarrow +
\end{align*}
\]

\[
\begin{align*}
\text{shape} &= \text{square} \land \text{color} = \text{blue} \rightarrow + \\
&\quad \Rightarrow \\
\text{shape} &= \text{quadrangle} \land \text{color} = \text{blue} \rightarrow +
\end{align*}
\]
Minimal Refinement Operators

- In many cases it is desirable, to only make minimal changes to a hypothesis
  - specialize only so much as is necessary to uncover a previously covered negative example
  - generalize only so much as is necessary to cover a previously uncovered positive example

- Minimal Generalization of a rule \( r \) relative to an example \( e \):
  - Find a generalization \( g \) of rule \( r \) and example \( e \) so that
    - \( g \) covers example \( e \) \((r \) did not cover \( e\))
    - there is no other rule \( g' \) so that \( e \leq g' < g \) and \( g' \geq r \)
  - need not be unique

- Minimal Specialization of a rule \( r \) relative to an example \( e \):
  - Analogously (specialize \( r \) so that it does not cover \( e \))
Minimal Generalization/Specialization

- least general generalization (lgg) of two rules
  - for Subset Generalization: the intersection of the conditions of the rules (or a rule and an example)

- most general specialization (mgs) of two rules
  - for Subset Generalization: the union of the conditions of the rules
Algorithm Find-S

I. $h = \text{most specific hypothesis in } H$
   
   (covering no examples)

II. for each training example $e$
   
   a) if $e$ is negative
      
      • do nothing
   
   b) if $e$ is positive
      
      • for each condition $c$ in $h$
         
         • if $c$ does not cover $e$
            
            • delete $c$ from $h$

III. return $h$

Note: when the first positive example is encountered, step II.b)
amounts to converting the example into a rule
(The most specific hypothesis can be written as a conjunction of all possible values of each attribute.)
Example

<table>
<thead>
<tr>
<th>No.</th>
<th>Sky</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Water</th>
<th>Forecast</th>
<th>sport?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>normal</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>rainy</td>
<td>cool</td>
<td>high</td>
<td>strong</td>
<td>warm</td>
<td>change</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>cool</td>
<td>change</td>
<td>yes</td>
</tr>
</tbody>
</table>

H₀: if false then +

if (sky = sunny & sky = rainy & ... & forecast = same & forecast = change) then +

<Ø,Ø,Ø,Ø,Ø,Ø>

H₁: <sunny, hot, normal, strong, warm, same>

H₂: <sunny, hot, ?, strong, warm, same>

H₃: <sunny, hot, ?, strong, warm, same>

H₄: <sunny, hot, ?, strong, ?, ?>

Short-hand notation:
- only body (head is +)
- one value per attribute
- Ø for false (full conjunction)
- ? for true (full disjunction)
Properties of Find-S

- **completeness:**
  - $h$ covers all positive examples

- **consistency:**
  - $h$ will not cover any negative training examples
  - but only if the hypothesis space contains a target concept (i.e., there is a single conjunctive rule that describes the target concept)

- **Properties:**
  - no way of knowing whether it has found the target concept (there might be more than one theory that are complete and consistent)
  - it only maintains one specific hypothesis (in other hypothesis languages there might be more than one)
  - Find-S prefers more specific hypotheses (hence the name) (it will never generalize unless forced by a training example)

Can we also find the most general hypothesis?
Algorithm Find-G

I. \( h = \text{most general hypothesis in } H \)  
   (covering all examples)

II. for each training example \( e \)
    a) if \( e \) is positive
       • do nothing
    b) if \( e \) is negative
       • for some condition \( c \) in \( e \)
          • if \( c \) is not part of \( h \)
             • add a condition that negates \( c \) 
               and covers all previous positive examples to \( h \)

III. return \( h \)
### Example

<table>
<thead>
<tr>
<th>No.</th>
<th>Sky</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Water</th>
<th>Forecast</th>
<th>sport?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>normal</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>rainy</td>
<td>cool</td>
<td>high</td>
<td>strong</td>
<td>warm</td>
<td>change</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>cool</td>
<td>change</td>
<td>yes</td>
</tr>
</tbody>
</table>

\( H_0: \) \textbf{if} true \textbf{ then } +

\[
\text{if} \ (\text{sky} = \text{sunny} \parallel \text{sky} = \text{rainy}) \ \& \ ... \ \& \ (\text{forecast} = \text{same} \parallel \text{forecast} = \text{change}) \textbf{ then } + \\
<?, ?, ?, ?, ?, ?>
\]

\( H_1: <?, ?, ?, ?, ?, ?> \)

\( H_2: <?, ?, ?, ?, ?, ?> \)

\( H_3: <?, ?, ?, ?, ?, \text{same}> \)

\( H_4: ???? \)

Other possibilities:
- \( <?, \ \text{hot}, ?, ?, ?, ?, ?> \)
- \( \text{<sunny}, ?, ?, ?, ?, ?> \)

There is no way to refine \( H_3 \) so that it covers example 4.
Uniqueness of Refinement Operators

- Subset Specialization is not unique
  - we could specialize any condition in the rule that currently covers the negative example
  - we could specialize it to any value other than the one that is used in the example
  → a wrong choice may lead to an impasse

- Possible Solutions:
  - more expressive hypothesis language (e.g., disjunctions of values)
  - backtracking
  - remember all possible specializations and remove bad ones later → Find-GSet algorithm

- Note: Generalization operators also need not be unique!
  - depends on the hypothesis language
Algorithm Find-GSet

I. \( h = \text{most general hypothesis in } H \) (covering all examples)
II. \( G = \{ h \} \)
III. for each training example \( e \)
   a) if \( e \) is positive
      - remove all \( h \in G \) that do not cover \( e \)
   b) if \( e \) is negative
      - for all hypotheses \( h \in G \) that cover \( e \)
        - \( G = G \setminus \{ h \} \)
        - for every condition \( c \) in \( e \) that is not part of \( h \)
          - for all conditions \( c' \) that negate \( c \)
            - \( h' = h \cup \{ c' \} \)
            - if \( h' \) covers all previous positive examples
              - \( G = G \cup \{ h' \} \)
IV. return \( G \)
Example

<table>
<thead>
<tr>
<th>No.</th>
<th>Sky</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Water</th>
<th>Forecast</th>
<th>sport?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>normal</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>rainy</td>
<td>cool</td>
<td>high</td>
<td>strong</td>
<td>warm</td>
<td>change</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>cool</td>
<td>change</td>
<td>yes</td>
</tr>
</tbody>
</table>

\[ G_0: \{ <?, ?, ?, ?, ?, ?> \} \]

\[ G_1: \{ <?, ?, ?, ?, ?, ?> \} \]

\[ G_2: \{ <?, ?, ?, ?, ?, ?> \} \]


We now have a set of hypotheses!

Remember all possible refinements that exclude example 3.
Correct Hypotheses

- **Find-GSet:**
  - finds *most general hypotheses* that are correct on the data
  - → has a bias towards general hypotheses

- **Find-SSet:**
  - can be defined analogously
  - finds *most specific hypotheses* that are correct on the data
  - → has a bias towards specific hypotheses

- Thus, the hypotheses found by Find-GSet or Find-SSet are not necessarily identical!

  → Could there be hypotheses that are correct but are neither found by Find-GSet nor by Find-SSet?
Version Space

- The version space in our example consists of 6 hypotheses
  - i.e. 6 rules are complete and consistent with the 4 seen examples

  \[
  \begin{align*}
  & <\text{sunny,}, ?, ?, \text{strong,}, ?, ?, > & <\text{sunny,}, \text{hot,}, ?, ?, ?, ?, > & <?, \text{hot,}, ?, \text{strong,}, ?, ?, > \\
  & & <\text{sunny,}, \text{hot,}, ?, ?, \text{strong,}, ?, ?, > \\
  S & <\text{sunny,}, \text{hot,}, ?, ?, \text{strong,}, ?, ?, > \\
  \end{align*}
  \]

- **Find-GSet** will find the rules in G
  - G are the most general rules in the version space

- **Find-SSet** will find the rules in S
  - S are the most specific rules in the version space
Version Space

- The Version Space $V$ is the set of all hypotheses that
  - cover all positive examples (*completeness*)
  - do not cover any negative examples (*consistency*)

- For structured hypothesis spaces there is an efficient representation consisting of
  - the **general boundary** $G$
    - all hypotheses in $V$ for which no generalization is in $V$
  - the **specific boundary** $S$
    - all hypotheses in $V$ for which no specialization is in $V$

- a hypothesis in $V$ that is neither in $G$ nor in $S$ is
  - a generalization of at least one hypothesis in $S$
  - a specialization of at least one hypothesis in $G$
Candidate Elimination Algorithm

- \( G = \) set of maximally general hypotheses
- \( S = \) set of maximally specific hypotheses

- For each training example \( e \)
  - if \( e \) is positive
    - For each hypothesis \( g \) in \( G \) that does not cover \( e \)
      - remove \( g \) from \( G \)
    - For each hypothesis \( s \) in \( S \) that does not cover \( e \)
      - remove \( s \) from \( S \)
    - \( S = S \cup \) all hypotheses \( h \) such that
      - \( h \) is a minimal generalization of \( s \)
      - \( h \) covers \( e \)
      - some hypothesis in \( G \) is more general than \( h \)
    - remove from \( S \) any hypothesis that is more general than another hypothesis in \( S \)
Candidate Elimination Algorithm (Ctd.)

- if $e$ is negative
  - For each hypothesis $s$ in $S$ that covers $e$
    - remove $s$ from $S$
  - For each hypothesis $g$ in $G$ that covers $e$
    - remove $g$ from $G$
    - $G = G \cup$ all hypotheses $h$ such that
      - $h$ is a minimal specialization of $g$
      - $h$ does not cover $e$
      - some hypothesis in $S$ is more specific than $h$
    - remove from $G$ any hypothesis that is less general than another hypothesis in $G$
### Example

<table>
<thead>
<tr>
<th>No.</th>
<th>Sky</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Water</th>
<th>Forecast</th>
<th>sport?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>normal</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>rainy</td>
<td>cool</td>
<td>high</td>
<td>strong</td>
<td>warm</td>
<td>change</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>cool</td>
<td>change</td>
<td>yes</td>
</tr>
</tbody>
</table>

\[
S_0: \{ <\varnothing,\varnothing,\varnothing,\varnothing,\varnothing,\varnothing> \} \\
G_0: \{ <?, ?, ?, ?, ?, ?> \}
\]

\[
S_1: \{ <\text{sunny}, \text{hot}, \text{normal}, \text{strong}, \text{warm}, \text{same}> \} \\
\]

\[
S_2: \{ <\text{sunny}, \text{hot}, ?, \text{strong}, \text{warm}, \text{same}> \} \\
\]

\[
S_3: \{ <\text{sunny}, \text{hot}, ?, \text{strong}, \text{warm}, \text{same}> \} \\
<?, \text{hot}, ?, ?, ?, ?> \\
<?, ?, ?, ?, ?, \text{same}> \}
\]

\[
S_4: \{ <\text{sunny}, \text{hot}, ?, \text{strong}, ?, ?> \} \\
G_4: \{ <\text{sunny}, ?, ?, ?, ?, ?> \} \\
<?, \text{hot}, ?, ?, ?, ?> \}
How to Classify these Examples?

- Version Space:


S  <sunny, hot, ?, strong, ?, ?>

- How to Classify these Examples?
How to Classify these Examples?

- Version Space:

  \[
  S \quad <\text{sunny}, \text{hot}, ?, \text{strong}, ?, ?>
  \]

- How to Classify these Examples?

<table>
<thead>
<tr>
<th>No.</th>
<th>Sky</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Water</th>
<th>Forecast</th>
<th>sport?</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>sunny</td>
<td>hot</td>
<td>normal</td>
<td>strong</td>
<td>cool</td>
<td>change</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>light</td>
<td>warm</td>
<td>same</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>sunny</td>
<td>hot</td>
<td>normal</td>
<td>light</td>
<td>warm</td>
<td>same</td>
<td>?</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
<td>maybe no</td>
</tr>
</tbody>
</table>
Properties

- Convergence towards target theory
  - convergence as soon as \( S = G \)
- Reliable classification with partially learned concepts
  - an example that matches all elements in \( S \) must be a member of the target concept
  - an example that matches no element in \( G \) cannot be a member of the target concept
  - examples that match parts of \( S \) and \( G \) are undecidable
- no need to remember examples (incremental learning)

Assumptions

- no errors in the training set
- the hypothesis space contains the target theory
- practical only if generality relation is (efficiently) computable
Other Generality Relations

- First-Order
  - generalize the arguments of each pair of literals of the same relation

- Hierarchical Values
  - generalization and specialization for individual attributes follows the ontology
Generalization Operators for Numerical Attributes

- Subset Generalization
  - generalization works as in symbolic case
  - specialization is difficult as there are infinitely many different values to specialize to

- Disjunctive Generalization
  - specialization and generalization as in symbolic case
  - problematic if no repetition of numeric values can be expected
    - generalization will only happen on training data
    - no new unseen examples are covered after a generalization

- Interval Generalization
  - the range of possible values is represented by an open or a closed interval
    - generalize by widening the interval to include the new point
    - specialize by shortening the interval to exclude the new point
Batch induction

- So far our algorithms looked at
  - all theories at the same time (implicitly through the version space)
  - and processed examples incrementally
- We can turn this around:
  - work on the theories incrementally
  - and process all examples at the same time
- Basic idea:
  - try to quickly find a complete and consistent rule
  - need not be in either $S$ or $G$ (but in the version space)
  
→ We can define an algorithm similar to FindG:
  - successively refine rule by adding conditions:
    - evaluate all refinements and pick the one that looks best
    - until the rule is consistent
Algorithm Batch-FindG

I. \( h = \text{most general hypothesis in } H \)
   \( C = \text{set of all possible conditions} \)

II. while \( h \) covers negative examples
    I. \( h_{\text{best}} = h \)
    II. for each possible condition \( c \in C \)
        a) \( h' = h \cup \{ c \} \)
        b) if \( h' \) covers
           • all positive examples
           • and fewer negative examples than \( h_{\text{best}} \)
           then \( h_{\text{best}} = h' \)

III. \( h = h_{\text{best}} \)

III. return \( h_{\text{best}} \)

Scan through all examples in database:
• count covered positives
• count covered negatives
Properties

- General-to-Specific (Top-Down) Search
  - similar to FindG:
    - FindG makes an arbitrary selection among possible refinements, taking the risk that it may lead to an inconsistency later
    - Batch-FindG selects next refinement based on all training examples
  - Heuristic algorithm
    - among all possible refinements, we select the one that leads to the fewest number of covered negatives
      - IDEA: the more negatives are excluded with the current condition, the less have to be excluded with subsequent conditions
  - Converges towards some theory in \( V \)
    - not necessarily towards a theory in \( G \)
  - Not very efficient, but quite flexible
    - criteria for selecting conditions could be exchanged