Lecture Notes for 6.824, Artificial Itelligence
 October, 1993

Inference Rules

A *Signature* is a set of constructor symbols each with an associated arity.

A *well formed expression* is

- a variable
- an application of a constructor of arity n to n expressions.

An *inference rule* consists of a set of antecedent expressions and a conclusion expression.

$$\begin{array}{l} A_1 \\ \vdots \\ A_2 \\ \hline C \end{array}$$

$\Sigma \ \vdash_R \ \Phi$ means Φ is provable from Σ using rules R.

Transitive Closure

```
(ARC ?X ?Y)
-----------------
(PATH ?X ?Y)
```

```
(ARC ?X ?Y)
(PATH ?Y ?Z)
-----------------
(PATH ?X ?Z)
```

$G \vdash_R$ `(PATH A B)` iff there is a path from A to B in the graph G.

A general rule compiler can convert the above rules to an $O(en)$ algorithm for an e edge n node graph.

Context Free Parsing

Assume a Chomsky normal form grammar with productions of the form

$$A \rightarrow BC$$

$$A \rightarrow c$$

Assume these productions are represented by assertions in the data base of the form

```
(PRODUCTION A B C)

(TERMINAL-PRODUCTION A c)
```

For nonterminal `A` and strings x and y, the formula (`REMOVE A` x y) means that removing a prefix of type A from x results in y.

```
(INPUT! (CONS ?X ?Y))
_____________________
(INPUT! ?Y)


(TERMINAL-PRODUCTION ?A ?T)
(INPUT! (CONS ?T ?X))
_____________________
(REMOVE ?A (CONS ?T ?X) ?X)


(PRODUCTION ?A ?B ?C)
(REMOVE ?B ?X ?Y)
(REMOVE ?C ?Y ?Z)
_____________________
(REMOVE ?A ?X ?Z)
```

$$\frac{\texttt{(REMOVE ?A ?X (NIL))}}{\texttt{(PARSES-AS ?X ?A)}}$$

$G,$`(INPUT! ?X)` $\vdash_R$ `(HAS-PARSE ?X ?A)` iff the string `?X` parses as nonterminal `?A` under the grammar G.

A general purpose rule compiler converts these rules to an $O(gn^3)$ algorithm where g is the number of productions in the grammar and n is the length of the input string.

The Binary Rule Transform

Define $B(R)$ as the result of applying the following transformation to each rule. In this transformation P_j represents a brand new predicate symbol and $x_{j,1}, \ldots, x_{j,k_j}$ are all the variables that appear both in one of $A_1, \ldots, A_j$ and in one of $A_{j+1}, \ldots, A_n$, C. Note that $B(R)$ has at most two antecedents per rule.

$$\frac{\begin{array}{l} A_1 \\ A_2 \\ \vdots \\ A_n \end{array}}{C}$$

beomes

$$\frac{A_1}{(P_1 \;\; x_{1,1} \;\ldots\; x_{1,k_1})}$$

$$\frac{\begin{array}{l} (P_1 \;\; x_{1,1} \;\ldots\; x_{1,k_1}) \\ A_2 \end{array}}{(P_2 \;\; x_{2,1} \;\ldots\; x_{2,k_2})}$$

$\vdots$

$(P_{n-1} \;\; x_{n-1,1} \;\; \ldots \;\; x_{1,k_{n-1}})$
A_n

C

Lemma: $\Sigma \;\; \vdash_R \;\; \Phi$ if and only if $\Sigma \;\; \vdash_{B(R)} \;\; \Phi$.

The Order Theorem

A *firing* of R in Γ is a ground instance of a rule in R all of whose antecedents are in Γ.

$C_R(\Sigma) \;\; = \;\; \{\Phi : \; \Sigma \;\; \vdash_R \;\; \Phi\}$

A *rule transition* of R on Σ is a firing of a rule in $B(R)$ in $C_{B(R)}(\Sigma)$.

Theorem: One can compute $C_R(\Sigma)$ in time proportional to the number of rule transitions of R on Σ.

There are $O(en)$ rule transitions in computing transitive closure of a graph with e edges and n nodes.

There are $O(gn^3)$ rule transitions involved in parsing a string of n symbols using a grammar with g productions.

Being Goal Directed (The Magic Sets Transformation)

A_1
A_2

C

becomes

(GOAL! C)

(GOAL! A_1)

(GOAL! C)
A_1

(GOAL! A_2)

(GOAL! C)
A_1
A_2

C

$\Sigma \vdash_R \Phi$ means $\Sigma \cup \{$(GOAL! Φ)$\} \vdash_R \Phi$

Lemma: $\Sigma \vdash_R \Phi$ iff $\Sigma \vdash_{M(B(R))} \Phi$

Theorem: The number of rule firings in $C_{B(M(B(R)))}(\Sigma \cup \{$(GOAL! Φ)$\})$ is bounded by a constant times the "backward chaining time" of the query $\Sigma \vdash_R \Phi$.

Turing Completeness

R is called *terminating* if $C_R(\Sigma)$ is finite for any finite Σ.

Theorem: Let $\mathcal{L}$ be any expression language over a finite signature. $\mathcal{L}$ is recursive if and only if $\mathcal{L}$ the language accepted by some terminating rule set.

A Programming Language that Defines P

A rule is *syntactically local* if every proper subexpression of the conclusion is a proper subexpression of some antecedent.

Local rules:

```
(SMALLER ?X ?Y)
(SMALLER ?Y ?Z)
---------------
(SMALLER ?X ?Z)
```

```
(= (CONS ?X ?Y) (CONS ?Z ?W))
-----------------------------
(= ?X ?Z)
```

Nonlocal rules:

```
(NATNUM ?X)
-----------------------------
(NATNUM (SUCCESSOR ?X))
```

```
(= ?X ?Y)
-----------------------------
(= (F ?X) (F ?Y))
```

Definiton: R *accepts* Φ if $\vdash_R \Phi$.

Theorem: An expression language over a finite signature is polynomial time decidable if and only if it is the language accepted by a syntactically local rule set.

Dijkstra Shortest Path

We introduce a special relation symbol $\leq$ whose second argument is always a number.

We now write Dijkstra Shortest path as follows.

```
(SOURCE! ?X)
(ARC ?X ?Z ?D)
-----------------------------
(≤ (DISTANCE ?X ?Z) ?D)
```

```
(SOURCE! ?X)
(≤ (DISTANCE ?X ?Y) ?D1)
(ARC ?Y ?Z ?D2)
-----------------------------
(≤ (DISTANCE ?X ?Z) (+ ?D1 ?D2))
```

A general rule compiler can convert these rules to an $e \log n$ where e is the number of arcs and n is the number of nodes (assuming that we are only given one souce).

The Inside/Outside Algorithm

Consider a stochastic grammar.

The production $A \xrightarrow{\frac{1}{3}} BC$ means that A goes to BC with probability $\frac{1}{3}$.

If the grammar contains $A \xrightarrow{\alpha} BC$ then we have the rule.

We can work with complexity rather than probability. ($\leq$ `(REMOVE ?A ?X ?Y) ?W`) means that it is possible to remove a string generated by nonterminal `?X` from the front of `?X` and get the string `?Y`, and furthermore there exists a derivation of the removed prefix such that the probability of that derivaaation is P and such that W is at least as large as $-\log_2 P$. In the following rules we assume that productions in the grammar are labeled with complexities rather than probabilities.

```
(INPUT! (CONS ?X ?Y))
---------------------
(INPUT! ?Y)
```

```
(TERMINAL-PRODUCTION ?A ?T ?W)
(INPUT! (CONS ?T ?X))
---------------------
(≤ (REMOVE ?A (CONS ?T ?X) ?X) ?W)
```

```
(PRODUCTION ?A ?B ?C ?W1)
(≤ (REMOVE-B ?X ?Y) ?W1)
(≤ (REMOVE-C ?Y ?Z) ?W2)
---------------------
(≤ (REMOVE-A ?X ?Z) (+ ?W1 ?W2 ?W3))
```

```
(≤ (REMOVE ?A ?X (NIL)) W
---------------------
(≤ (PARSES-AS ?X ?A) W)
```

G, `(INPUT! ?X)` $\vdash_R$ ($\leq$ `(HAS-PARSE ?X ?A)` W) iff A generates the string X with nonzero probability and W is $-\log_2 P$ where P the probability of the most likely derivation of X from A.

A general purpose rule compiler converts these rules to an $O(gn^3 \log(gn))$ algorithm where g is the number of productions in the grammar and n is the length of the input string.

Order Theorem for Bounds

Definition: A formula $(\leq \; s \; b)$ will be called a *redundant* element of Γ if Γ contains some formula of the form $(\leq \; s \; b')$ for $b' < b$.

Definition: Let $C'_R(\Sigma)$ be the nonredundant elements of $C_R(\Sigma)$.

Theorem: Provided that no conclusion bound is smaller than any antecedent bound in any given rule firing, we can compute $C'_R(\Sigma)$ in $c \log b$ time where c is the number of transitions of R on Σ and b is the number of bound assertions in $C'_R(\Sigma)$.

There are $O(e)$ transitions and $O(n)$ bounds in computing Dijkstra shortest path.

There are $O(gn^3)$ transitions and $O(gn^2)$ bounds in computing the most likely parse.