Lecture Notes for 6.824, Artificial Itelligence
©1993 David McAllester, all rights reserved 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 assertion (REMOVE A x y) means that removing a prefix of type A from x results in y.

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

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

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

```
(REMOVE ?A ?X (NIL))
--------------------
(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 Order Theorem

Let Σ be a data base of premises and let R be a set of rules.

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

A *prefix firing* of a rule

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

in the set Γ is a ground instance $A'_1, \ldots A'_j$ of the antecedents $A_1, \ldots, A_j$ where $j \leq n$ such that each A'_i is a member of Γ. Typically Γ is $C_R(\Sigma)$.

For example, consider the following rule.

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

The pair of assertions
`(PRODUCTION S NP VP)`

```
(REMOVE NP
        (CONS JOHN (CONS RAN NIL))
        (CONS RAN NIL))
```

is a prefix firing of this rule in the set $C_R(\Sigma)$ provided these two assertions are both members of $C_R(\Sigma)$.

Theorem: Assuming that hash table operations take unit time one can compute $C_R(\Sigma)$ in time proportional to the number of prefix firings of rules in R in the set $C_R(\Sigma)$.

There are $O(en)$ prefix firings in computing transitive closure of a graph with e edges and n nodes. To see this consider the following rule.

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

There are e prefix firings involving just the first antecedent. There are at most en firings involving both antecedents. Therefore the total number of prefix firings of this rule is $O(en)$. This rule has more prefix firings than any other rule in the rule set for transitive closure. Hence the total number of prefix firings is $O(en)$.

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

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 $x_{j,1}$, ..., x_{j,k_j} are all the variables that appear both in one of A_1, ..., A_j and in one of A_{j+1}, ..., A_n, C. Note that $B(R)$ has at most two antecedents per rule.

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

becomes

$$\begin{array}{l} A_1 \\ \hline (P_1\ \ x_{1,1}\ \ \ldots\ \ x_{1,k_1}) \end{array}$$

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

$\vdots$

$$\begin{array}{l} A_n \\ (P_{n-1}\ \ x_{n-1,1}\ \ \ldots\ \ x_{1,k_{n-1}}) \\ \hline C \end{array}$$

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

Being Goal Directed (The Magic Sets Transformation)

Let $M(R)$ be the set of rules derived from R by replacing each rule

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

in R by the rules

$$\begin{array}{l} (\texttt{GOAL!}\ C) \\ \hline (\texttt{GOAL!}\ A_1) \end{array}$$

$$
\begin{array}{l}
\texttt{(GOAL! } C\texttt{)} \\
A_1 \\
\hline
\texttt{(GOAL! } A_2\texttt{)}
\end{array}
$$

$$
\begin{array}{l}
\texttt{(GOAL! } C\texttt{)} \\
A_1 \\
A_2 \\
\hline
C
\end{array}
$$

$\Sigma \mathrel{\vdash\!\!\!|}_R \Phi$ means $\Sigma \cup \{\texttt{(GOAL! }\Phi\texttt{)}\} \vdash_R \Phi$

Lemma: $\Sigma \vdash_R \Phi$ iff $\Sigma \mathrel{\vdash\!\!\!|}_{M(B(R))} \Phi$

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

Turing Completeness

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

Definition: R *accepts* Φ if $\mathrel{\vdash\!\!\!|}_R \Phi$. ($\mathrel{\vdash\!\!\!|}_R \Phi$ is shorthand for $\emptyset \mathrel{\vdash\!\!\!|}_R \Phi$ where $\emptyset$ is the empty set.)

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:

$$
\begin{array}{l}
\texttt{(SMALLER ?X ?Y)} \\
\texttt{(SMALLER ?Y ?Z)} \\
\hline
\texttt{(SMALLER ?X ?Z)}
\end{array}
$$

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

Nonlocal rules:

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

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

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 $e \log n$ algorithm where e is the number of arcs and n is the number of nodes (assuming that we are only given one source).

The Inside/Outside Algorithm

Consider a stochastic grammar.

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

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 derivation 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 *bound assertion* is a formula of the form $(\leq\ e\ \ b)$.

Definition: A bound assertion $(\leq\ s\ \ b)$ will be called *redundant* in $C_R(\Sigma)$ if $C_R(\Sigma)$ contains a bound formula $(\leq\ s\ \ b')$ giving a tighter bound for s, i.e., where $b' < b$.

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

Theorem: Assuming that hash table operations take unit time, that no conclusion bound is smaller than any antecedent bound in any given rule firing, and that any rule with a bound formula antecedent has a bound formula conclusion, we can compute $C'_R(\Sigma)$ in $c + b \log b$ time where c is the number of prefix firings of R in $C'_r(\Sigma)$ and b is the number of bound assertions in $C'_R(\Sigma)$.

There are $O(e)$ prefix firings and $O(n)$ bound formulas in computing Dijkstra shortest path.

There are $O(gn^3)$ prefix firings and $O(gn^2)$ bound formulas in computing the most likely parse.

1 Historical Comments

The basic idea of using forward chaining inference rules as a programming language has a long history. It was used in early AI languages such as AMORD (Sussman et al. circa 1978). It underlies the OPS5 programming language for AI expert systems. The order theorem presented in these notes, which allows the order of run time of a rule set to be easily determined, is original and has not appeared elsewhere. However, the proof of the theorem (which is not given here) involves methods similar to those used in the Rete matching algorithm for OPS5. The magic sets transformation was developed in the database community. An overview of this and related techniques can be found in the chapter "Bottom-UP Evaluation of Logic Programs" by Jeffery Naughton and Raghu Ramakrishnan in the book "Computational Logic: Essays in Honor of Alan Robinson" edited by Lassez and Plotkin, MIT Press. The bounds predicate used in these notes is similar to constructions introduced by Zaniolo and others and presented in PODS (Principles of Database systems), 1992. However, the order theorem for bounds assertions is original here. The characterization of the complexity class P as the class of term languages recognized by syntactically local rule sets was presented in the paper "New Results on Local Inference Relations" by Bob Givan and David McAllester in KR92 (Principles of Knowledge Representation). This result is similar to earlier results on Descriptive complexity theory by Neil Imerman and Moshe Vardi.