Statistical Binary Parsing

Using Machine Learning to Extract Code from Uncooperative Programs

Nathan Rosenblum

Paradyn Project

Paradyn / Condor Week

Madison, Wisconsin

April 30 – May 3, 2007

Research Participants

- Barton Miller - UW Madison
- Jerry Zhu - UW Madison
- Karen Hunt - DoD
- Jeff Hollingsworth - UMD

Context of Current Work

- Exploratory
- Focus: evaluating machine learning techniques
- Eventual integration with Dyninst

Exploration of machine learning techniques → Selection & optimization of best methods → Integration into Dyninst tool

Current | Phase 2 | Phase 3

Talk Outline

- Binary parsing challenges
- Machine Learning Infrastructure
- Testing and Evaluation Infrastructure
- Preliminary Results

Automated Batch Parsing

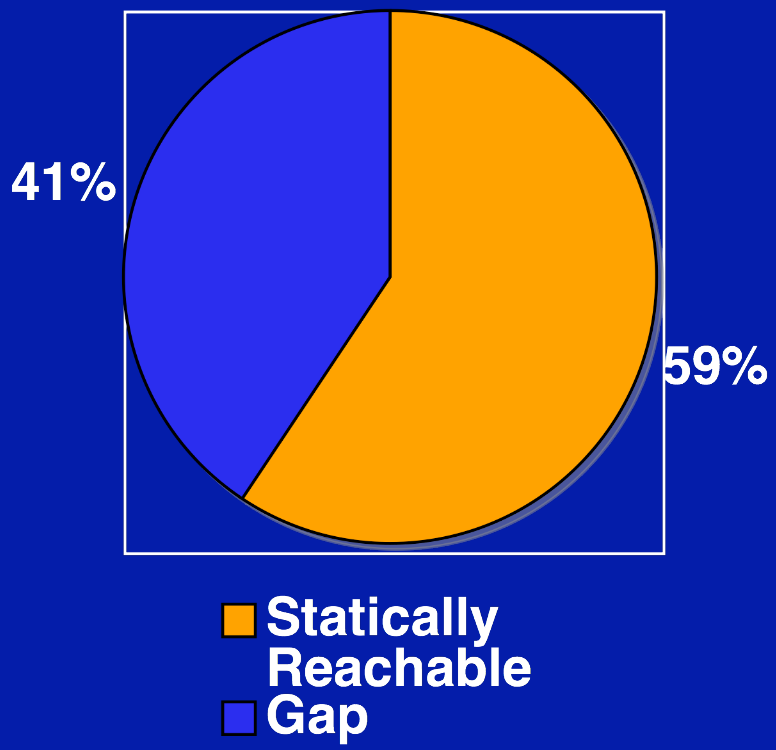
- Cannot rely on human input
  - Parsing very large (100 MB) binaries
  - Parsing large numbers of binaries
  - Decisions require expert knowledge
- Complete & accurate information is essential
  - Binary modification, instrumentation
  - Misidentifying code can have catastrophic consequences
- Goal: Find code location in binaries
  - Eliminate *false positives*
  - Minimize *false negatives*

Parsing Challenges

- Obtaining full coverage may be difficult:
  - Missing symbol information
  - Variability in function layout (e.g. code sharing, outlined basic blocks)
  - High degree of indirect control flow
- Basic strategy: recursive descent parsing
  - Disassemble from known entry points
  - Discover functions through calls

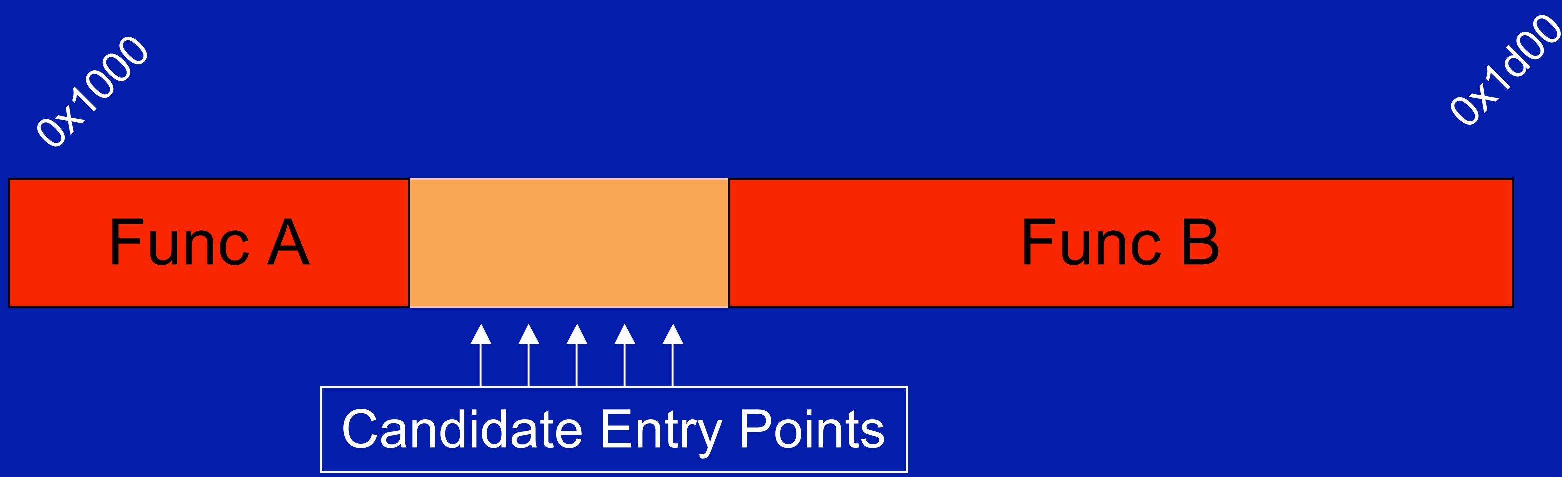
Incomplete Parsing Coverage

- 41% of functions in surveyed binaries unreachable
- As many as 90% in some programs
- Unreachable functions occupy *gap regions* in the binary



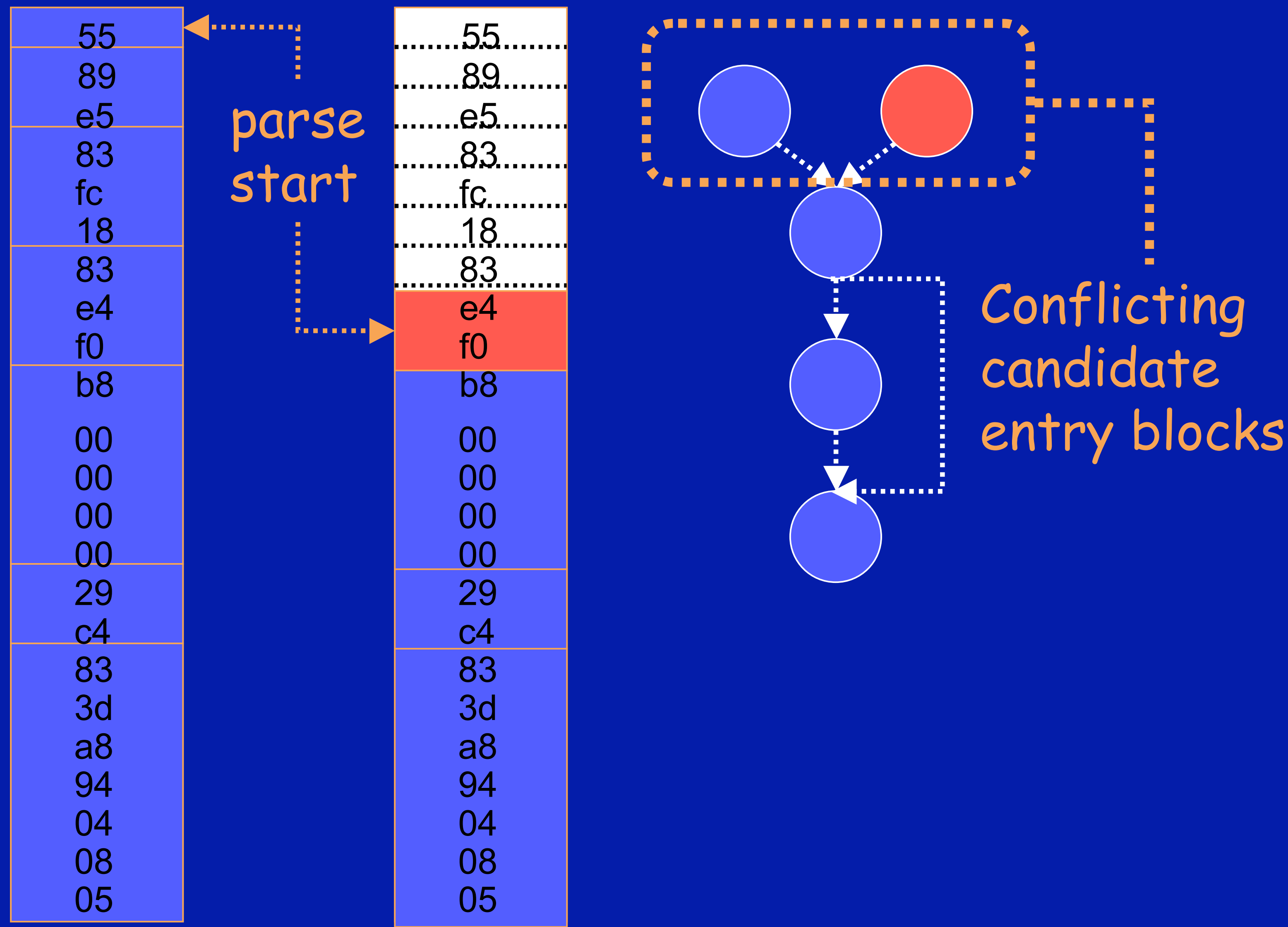
Challenge: Accurate Gap Parsing

- Gaps are sequences of bytes
- Need to identify functions in gaps
  - Equivalently, identify function entry blocks



Offset Parsing Alignment

```
push ebp
mov esp,
    ebp
sub 0x18
    esp
and 0xf0
    esp
mov 0x0
    eax
sub eax,
    esp
cmpl 0x5
0x80494a8
```



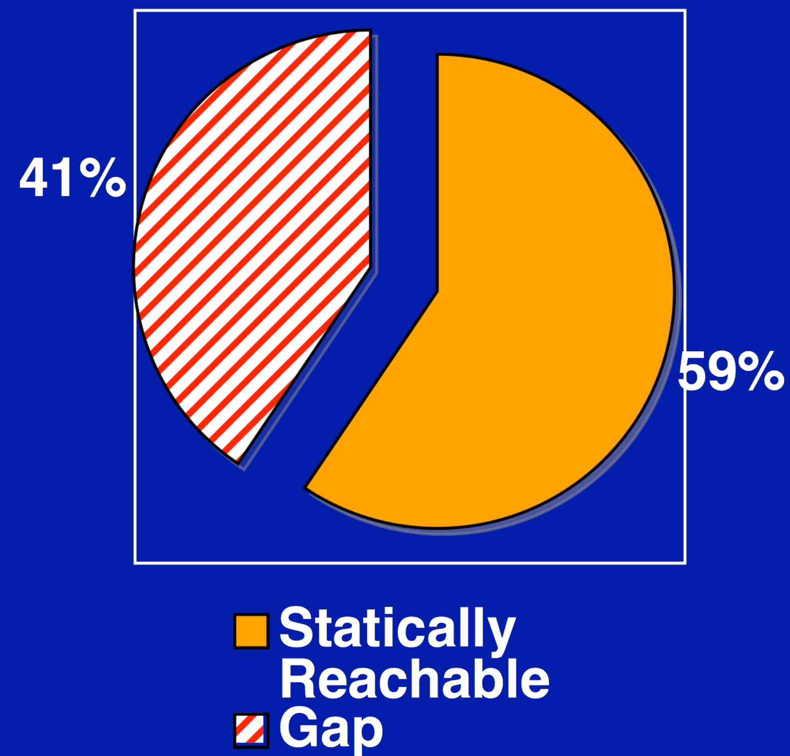
Current Dyninst Techniques

- Dyninst searches for common patterns
  - `push %ebp; mov %esp,%ebp`
  - `push %esi; mov %esi,<mem>`
- Performs well
  - Low false positive rate: 92% precision on average
- Heuristic - patterns are moving target
- Larger programs - more false positives
- Compiler may not emit expected preamble
  - Partial known sequences

Exploiting Available Information

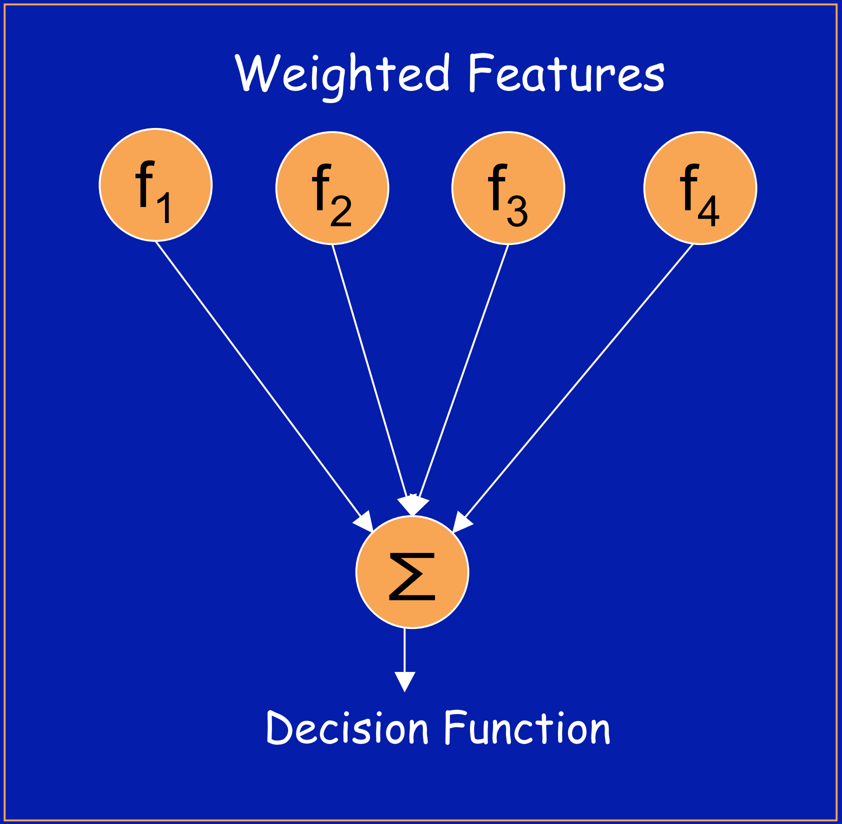
- *Some* properties of functions are relatively uniform
  - E.g., stack setup

- Use properties of known code to search gaps



Statistical Binary Parsing

- Parsing as a supervised machine-learning problem
  - Build model from training examples
  - Use model to classify code in gaps
- Goals:
  - Extensible: incorporate multiple *features*
  - Opportunistic: exploit all available information



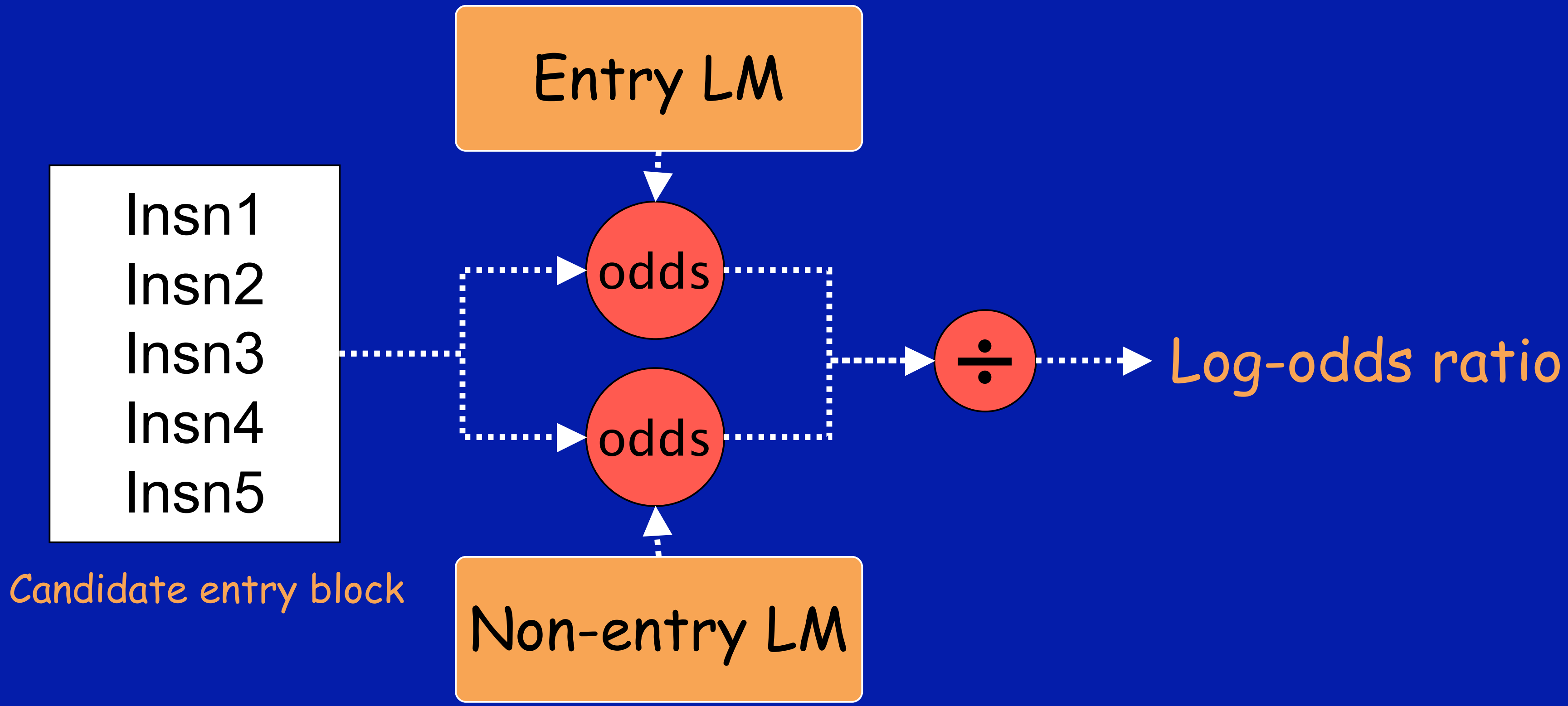
A binary classifier for candidate entry blocks

Learning Infrastructure

- Logistic Regression classifier
- Incorporates several features:
  - Instruction frequency (language models)
  - Function entry sequences
  - Control flow
- Assigns probability to candidate functions

Language Models

- Frequency of instruction occurrence
- Compares entry and non-entry models



Function Entry Sequences

- Method 1: Maximum Prefix Match Length
  - Incorporates instruction ordering
  - Construct *prefix trie* of entry block sequences
  - Compute maximum match length for candidate entry blocks

Candidate 1: actual entry block

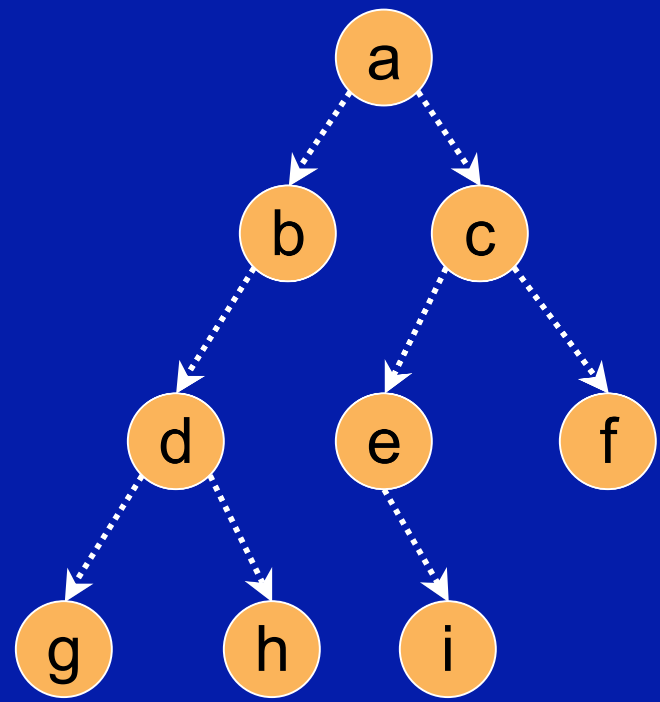
`a,b,d,h,x,…` MPML: 4

Candidate 2: non-entry block

`a,q,x,y,z,…` MPML: 1

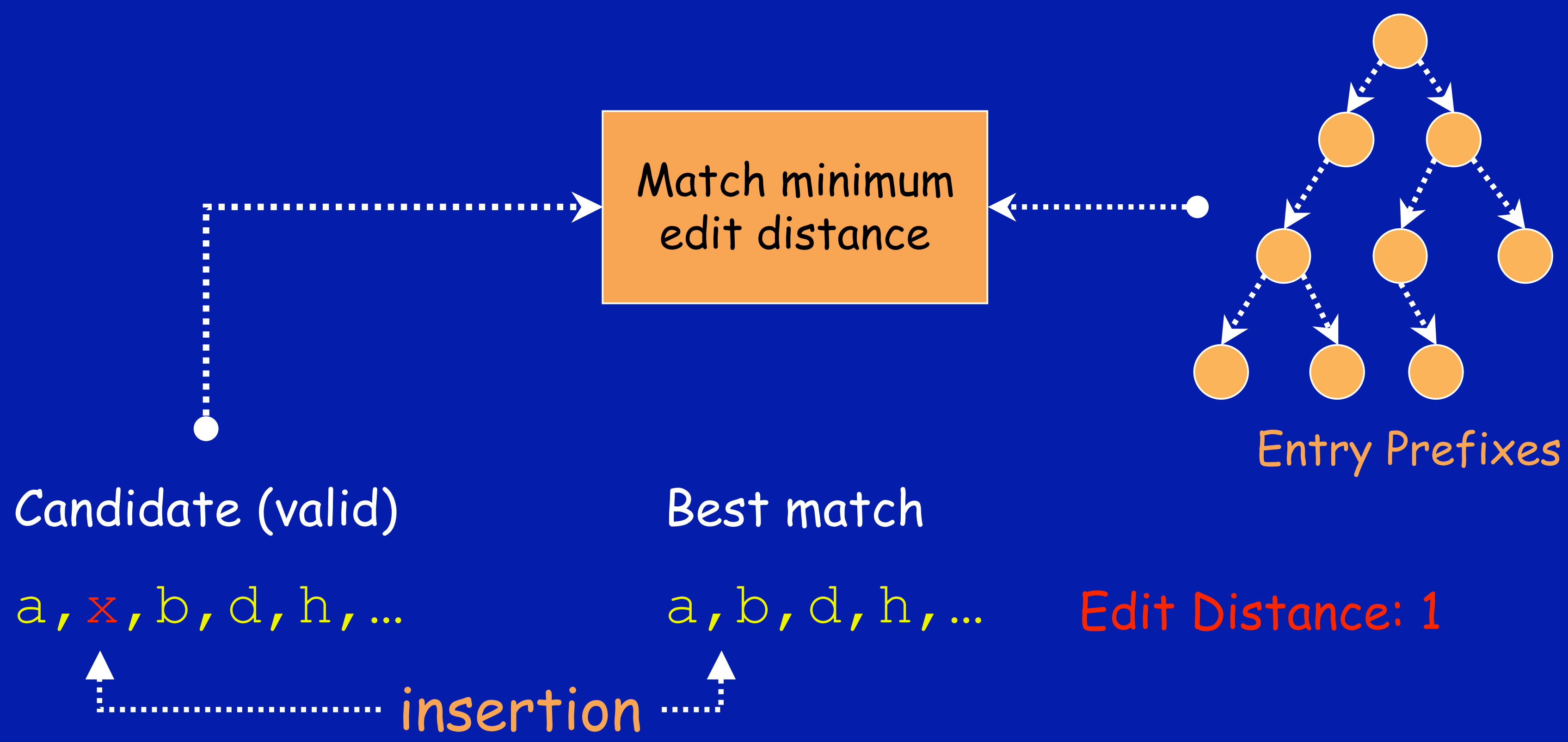
Limited flexibility!

`a,x,b,d,h,…` MPML: 1

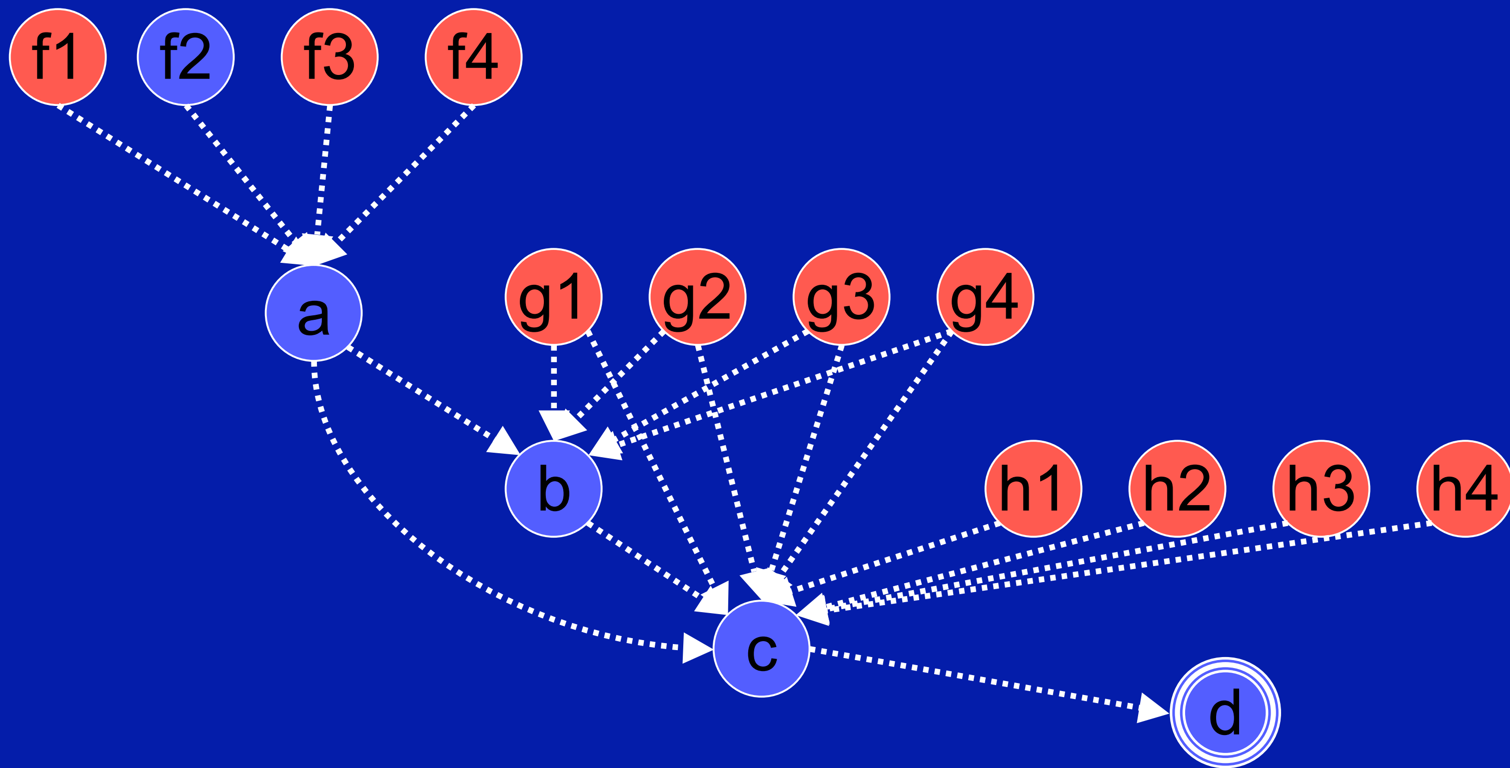


Function Entry Sequences

- Method 2: Fuzzy String Matching
  - Levenshtein Distance counts edits between strings
    - Insertion, deletion, change
  - Flexible: matches sequences but allows gaps



Incorporating Control Flow



Parsing from every byte in a range creates a graph

$$\text{Reachability Ratio} = \frac{\text{\# blocks reachable from candidate}}{\text{\# blocks connected to candidate}}$$

Experimental Framework

- Goal: evaluate effectiveness of features
- 625 Linux x86 binaries
- Binaries have full symbol tables
  - Function locations provide ground truth *reference set*
- Stripped binaries provide training data
- Dyninst prefix heuristic provides baseline

Obtaining Training and Test Data

- Classifier is trained and evaluated on each binary *independently*
- Positive training examples:
  - Known function entry blocks
- Negative training examples:
  - Known non-entry blocks
  - Blocks generated from parse at every byte within known functions ("anti-gaps")
- Test examples are all candidates in gaps

Scaling Experiments

- Experiment design facilitates scaling
  - Separation of model creation, training, and evaluation
  - Independent analysis of each binary
  - Suitable for batch processing systems like Condor

- Reduced cost in final Dyninst implementation
  - Early rejection of invalid parses
  - On-demand analysis of sub-regions of gaps
  - Final approach will use subset of techniques

Results

- Language Model features have limited utility
  - Limited training data
  - May be improved by training over whole corpus
- Prefix-based features work well
  - LD better than MPML
  - LD is current best combined with Dyninst heuristic
  - Most sensitivity to training data variation
- Incorporating control flow is essential
  - 60% reduction in false positives over best method alone

Results

- Current status:
  - 70% reduction in false positives over Dyninst heuristic
  - Nearly identical false negative rates

Prog	Total Functions	Gap Funcs	Precision	Recall
grep	140	94	100%	90.5%
mutt	1122	223	98.6%	98.6%
emacs	3214	1596	99.9%	99.9%
Abiword	13844	538	100%	100%
gpg	991	172	41.7%	99.4%

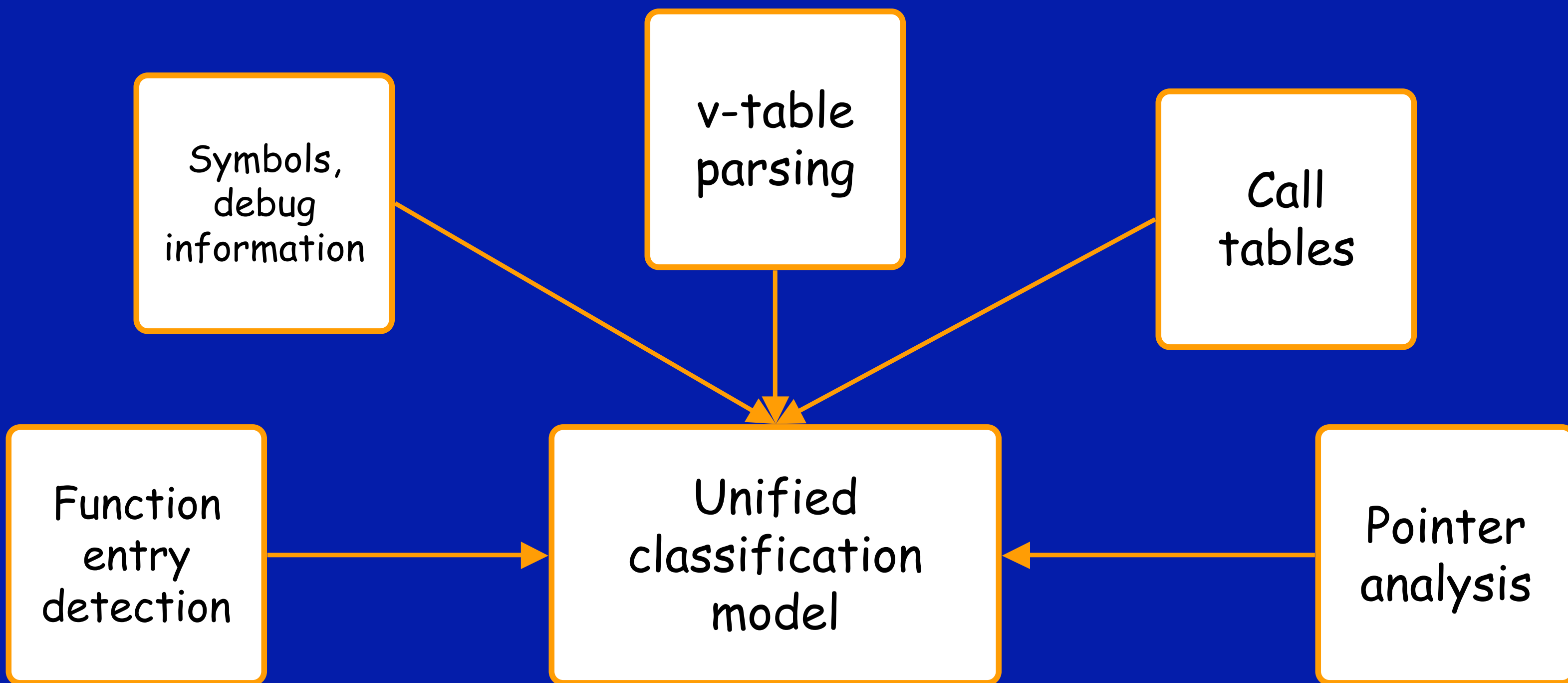
Future Work

- Model extension, evaluation and refinement
  - What other features characterize entry points?
  - Which features best distinguish valid entry points?

- Integration into Dyninst
  - Model training
  - Parsing optimizations
  - API extensions
  - Fall 2007

Future Work

- Dealing with limited training data
  - Can similar binaries be exploited to obtain more training examples?
- Incorporating additional sources of information



Questions?

Backup slides

Language Models

- Obtained by Maximum Likelihood Estimate (MLE) of instructions (unigram) and pairs of instructions (bigram)

Probabilities based on frequency of instruction occurrence

$$P(insn_k) = \frac{\sum_{b \in EntryBlocks} cnt_b(insn_k) + 1}{\sum_{b \in EntryBlocks} \sum_{i \in Insns} cnt_b(i) + |Insns|}$$

$$P(block_k) = \prod_{i \in Insns_b} P(i)$$

Language Models

- Log-odds ratio computed from language models

- Two models trained:
  - Entry blocks
  - Non-entry blocks

$$odds_{entry}(b) = \frac{P_{entry}(b)}{1 - P_{entry}(b)}$$

$$odds_{nonentry}(b) = \frac{P_{nonentry}(b)}{1 - P_{nonentry}(b)}$$

$$\mathrm{LOR}(b) = \log\left(\frac{odds_{entry}(b)}{odds_{nonentry}(b)}\right)$$

An example