

# *ESPRESSO/FASTAR'05*

## Characterization of Finite Automata Implementations: A Preliminary Taxonomy

E. Ketcha Ngassam

Bruce W. Watson

Derrick G. Kourie

## *Agenda*

- Background and Motivations
- Characterization of Finite Automata
- Characterization of Acceptors
- Constraints Identification: Refining the Characterization
- Deriving the kwk Algorithms
- A Preliminary Taxonomy Tree
- Future Work

## Background and Motivations

- Two Finite Automata Implementation Approaches
  - Conventional: Table-driven Algorithm
    - Memory load problems (transition matrix)
    - Limited capacity of data cache
  - Alternative: Hardcoded Algorithm
    - Regular expressions (Thompson-1968)
    - Pattern Matching (Knuth -1977)
    - Instructions load problems
    - Limited capacity of instruction cache

## Background and Motivations

- Need to explore other strategies in order to:
  - To reduce memory load
    - Dynamic data (states) allocation in memory
  - To reduce instruction load
    - Dynamic instructions (states) allocation in memory
  - To exploit hardware's capabilities:
    - Cache structure and principles
      - Cache mapping and replacement principles
      - Cache locality of reference
      - Unified cache / Split cache

## Characterization of Finite Automata

- A Finite Automaton  $M$  is a 5-tuple  $(s_0, Q, F, \Sigma, \delta)$  where:
  - $s_0$  is the initial state
  - $Q$  is the set of states ( $s_0 \in Q$ )
  - $F$  the set of final states ( $F \subseteq Q$ )
  - $\Sigma$  the set of alphabet symbols
  - $\delta$  the transition function

## Characterization of Finite Automata

- $\delta: Q \times \Sigma \rightarrow Q$   
 $\delta(q_i, c) = q_j$
- Define the transition set  $\Delta \subseteq Q \times \Sigma \times Q$ 
  - Elements of  $\Delta$  are triplets of the form  $(q_i, c, q_j)$
  - $\Delta \in \mathbf{P}(Q \times \Sigma \times Q)$ : power set of  $Q \times \Sigma \times Q$
  - $|Q|$  = number of states of  $M$
- Define  $\Sigma^*$ , the set of all strings of  $\Sigma$ 
  - $\mathbf{L}(M)$  is the set of all strings accepted by  $M$
  - $\mathbf{L}(M) \subseteq \Sigma^*$
  - $S \in \mathbf{L}(M) \Leftrightarrow S \in \Sigma^*$

## Characterization of Acceptors

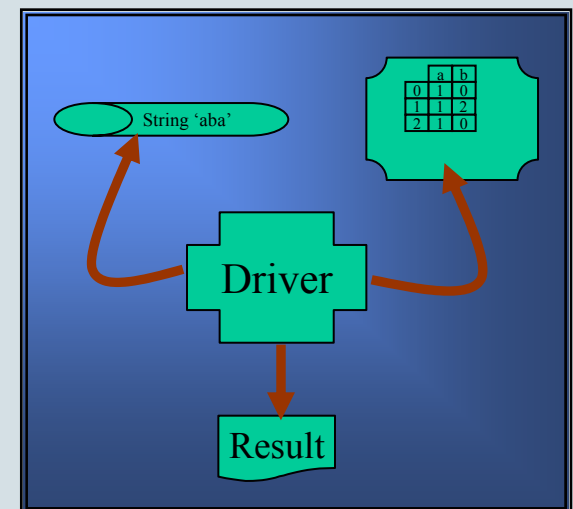
- An acceptor is a string recognizer that uses FA
- Define  $B = \{\text{true}, \text{false}\}$  the Boolean set;  $b \in B$ , a Boolean value
- $\mathcal{R}((Q \times \Sigma \times Q) \times \Sigma^* \rightarrow B)$   
 $\mathcal{R}(\Delta, S) = b$ 
  - $\mathcal{R}(\Delta, S) = \text{true} \Leftrightarrow S \in \mathbf{L}(M)$
  - $\mathcal{R}(\Delta, S) = \text{false} \Leftrightarrow S \notin \mathbf{L}(M)$

# Characterization of FAs Implementations

## Characterization of Acceptors:

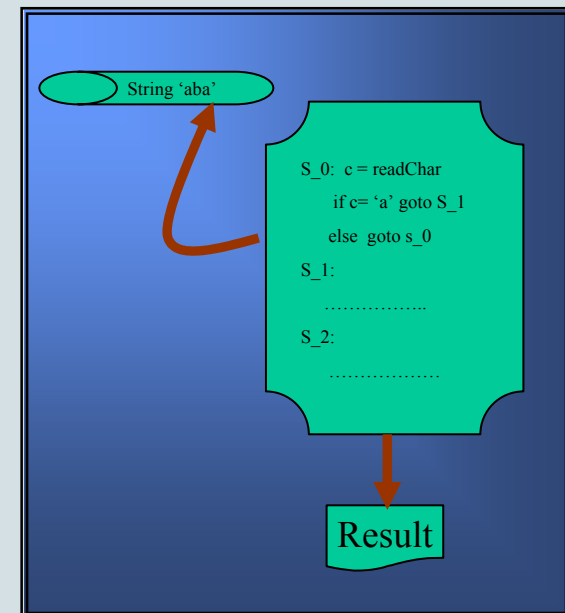
### Implementation of $\mathbf{r}(\Delta, S)$

- **Table driven:** Driver + Transition matrix
  - $\Delta$  implemented as  $\Delta_t$
  - $\mathbf{r}(\Delta, S) \equiv \mathbf{r}(\Delta_t, S)$



## Characterization of Acceptors: Implementation of $\mathbf{r}(\Delta, S)$

- **Hardcoded:** Instructions only
  - $\Delta$  implemented as  $\Delta_h$
  - $\mathbf{r}(\Delta, S) \equiv \mathbf{r}(\Delta_h, S)$

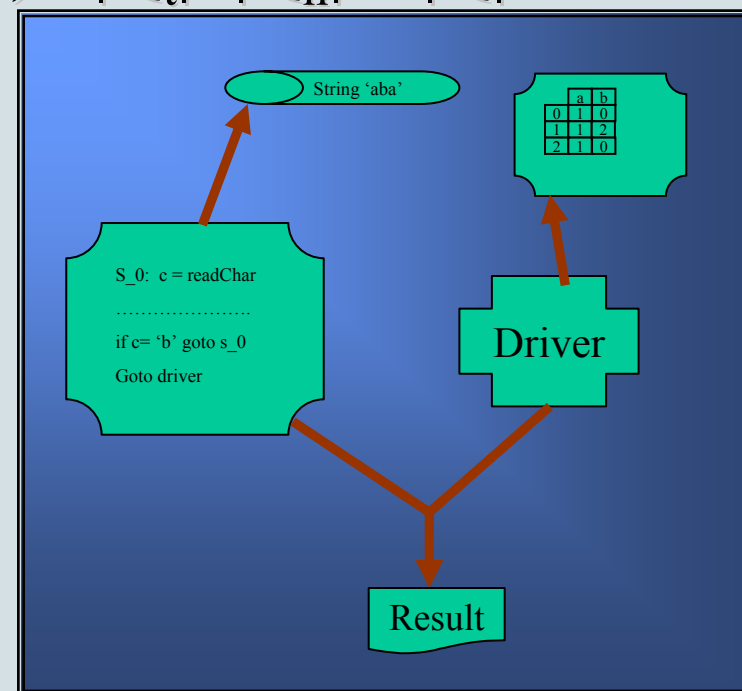


# Characterization of FAs Implementations

## Characterization of Acceptors:

### Implementation of $r(\Delta, S)$

- **Mixed-Mode: Driver+Matrix +Instructions**
  - $r(\Delta_t \cup \Delta_h, S) \equiv r'(\Delta_t, \Delta_h, S) : |Q_t| + |Q_h| = |Q|$



## Constraints Identification: Refining the Characterization

- Consider  $\mathbf{r}(\Delta_t, \Delta_h, S)$ :  $|Q_t| + |Q_h| = |Q|$ 
  - This approach is somewhat general
  - It can be specialized in terms of constraints on
    - States
      - How many are hardcoded?
      - How many are in table?
      - How states are organized?
      - Etc
    - Provides various implementation strategies
      - Strategies or constraints become part of the characterization

## Constraints Identification: Dynamic Auxiliary Automaton ( $C_1$ )

- Acceptance is made using a Dynamic Auxiliary Automaton
  - Created at runtime
  - Function to the string path
  - States renamed in a new memory location
- $C_1$  is a Boolean in the characterization
  - $C_{1t}$ : Dynamic Auxiliary Table
  - $C_{1h}$ : Dynamic Auxiliary Hardcode
- New Characterization:  $\mathbf{r}_1(\Delta_t, \Delta_h, S, C_{1t}, C_{1h}): |Q_t| + |Q_h| = |Q|$ 
  - TD:  $\mathbf{r}_1(\Delta_t, \phi, S, t, f): |Q_t| = |Q|$
  - HC:  $\mathbf{r}_1(\phi, \Delta_h, S, f, t): |Q_h| = |Q|$
- TD and HC use  $C_1$  independently in mixed-mode

## Constraints Identification: State Ordering ( $C_2$ )

- States are organized on a predefined order before acceptance testing
  - Done before acceptance testing
  - Uses a profiler to define the ordering policy
    - e.g: Frequently used states are put together
- $C_2$  is a Boolean in the characterization
  - $C_{2t}$ : States in the table are ordered
  - $C_{2h}$ : Hardcoded states are ordered
- New Characterization:  $\mathbf{r}_2(\Delta_t, \Delta_h, S, C_{2t}, C_{2h}) : |Q_t| + |Q_h| = |Q|$ 
  - TD:  $\mathbf{r}_2(\Delta_t, \phi, S, t, f) |Q_t| = |Q|$
  - HC :  $\mathbf{r}_2(\phi, \Delta_h, S, f, t) |Q_h| = |Q|$
- TD and HC use  $C_2$  independently in Mixed-mode

## Constraints Identification: State Swapping ( $C_3$ )

- States are swapped between one another
  - Done at runtime
  - Uses a policy such as LRU to interchange states position
- $C_3$  is a Boolean in the characterization
  - $C_{3t}$ : Table-driven states are swapped
  - $C_{3h}$ : hardcoded states are swapped
- New Characterization:  $\mathbf{r}_3(\Delta_t, \Delta_h, S, C_{3t}, C_{3h}): |Q_t| + |Q_h| = |Q|$ 
  - TD:  $\mathbf{r}_3(\Delta_t, \phi, S, t, f): |Q_t| = |Q|$
  - HC:  $\mathbf{r}_3(\phi, \Delta_h, S, f, t): |Q_h| = |Q|$
- TD and HC use  $C_3$  independently in Mixed-mode

## Constraints Identification: State Caching ( $C_4$ )

- A space in memory referred to as “cache” is predefined with a number of states
  - Space allocated with states before acceptance testing
  - Acceptance testing only happens within the predefined “cache”
  - Uses a policy such as LRU to allocate new states in “cache”
- $C_4$  is a natural number:  $C_4 \leq |Q|$ 
  - $C_4 = 0$  means there is no states caching
  - $C_{4t}$ : Table-driven states use a predefined “Cache”
  - $C_{4h}$ : Hardcoded states use a predefined “Cache”
- New Characterization:  $\mathbf{r}_4(\Delta_t, \Delta_h, S, C_{4t}, C_{4h})$ :  $|Q_t| + |Q_h| = |Q|$ 
  - TD:  $\mathbf{r}_4(\Delta_t, \phi, S, m, 0)$   $|Q_t| = |Q|$ ,  $m \leq |Q|$
  - HC:  $\mathbf{r}_4(\phi, \Delta_h, S, 0, m)$   $|Q_h| = |Q|$ ,  $m \leq |Q|$
- TD and HC use  $C_4$  independently in Mixed-mode;  $C_{4t} + C_{4h} \leq |Q|$
- If  $C_5$  is set and  $C_4 = C_5$ , then Dynamic allocation occurs within the “cache”

## Constraints Identification: Dynamic State Allocation ( $C_5$ )

- States are allocated dynamically in memory at runtime before acceptance testing
  - This is a JIT like processing of acceptors
  - Acceptance testing only happens within the dynamically allocated space
  - States are not renamed in the newly allocated space
  - State allocation depends on the string path
- $C_5$  is a natural number:  $C_5 \leq |Q|$ 
  - $C_5 = 0$  means there is no states caching
  - $C_{5t}$ : Table-driven states are dynamically allocated during acceptance testing
  - $C_{5h}$ : Hardcoded states are dynamically allocated during acceptance testing
- New Characterization:  $\mathbf{r}_4(\Delta_t, \Delta_h, S, C_{5t}, C_{5h})$ :  $|Q_t| + |Q_h| = |Q|$ 
  - TD:  $\mathbf{r}_4(\Delta_t, \phi, S, m, 0)$   $|Q_t| = |Q|$ ,  $m \leq |Q|$
  - HC:  $\mathbf{r}_4(\phi, \Delta_h, S, 0, m)$   $|Q_h| = |Q|$ ,  $m \leq |Q|$
- TD and HC use  $C_4$  independently in Mixed-mode;  $C_{4t} + C_{4h} \leq |Q|$
- If  $C_4$  is set and  $C_4 = C_5$ , then Dynamic allocation occurs within the “cache”

## Constraints Identification: Generalization

- Without loss of generality, we define

$$\mathbf{r}(\Delta_t, \Delta_h, S, C_{1t}, C_{1h}, C_{2t}, C_{2h}, C_{3t}, C_{3h}, C_{4t}, C_{4h}, C_{5t}, C_{5h})$$

$$|Q_t| + |Q_h| = |Q| ; C_{4t} + C_{4h} \leq |Q| ; C_{5t} + C_{5h} \leq |Q|$$

- Many algorithms can be derived
  - Some may not be useful in practice

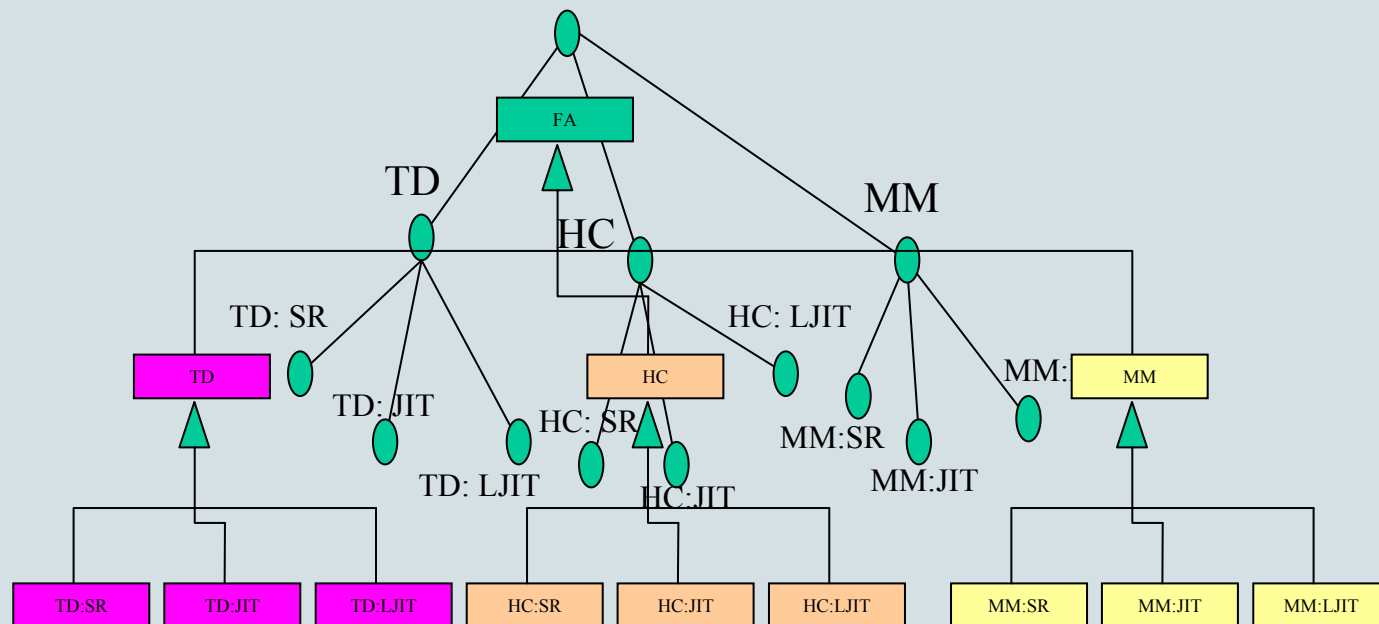
## Deriving the kwk Algorithms

kwk algorithms

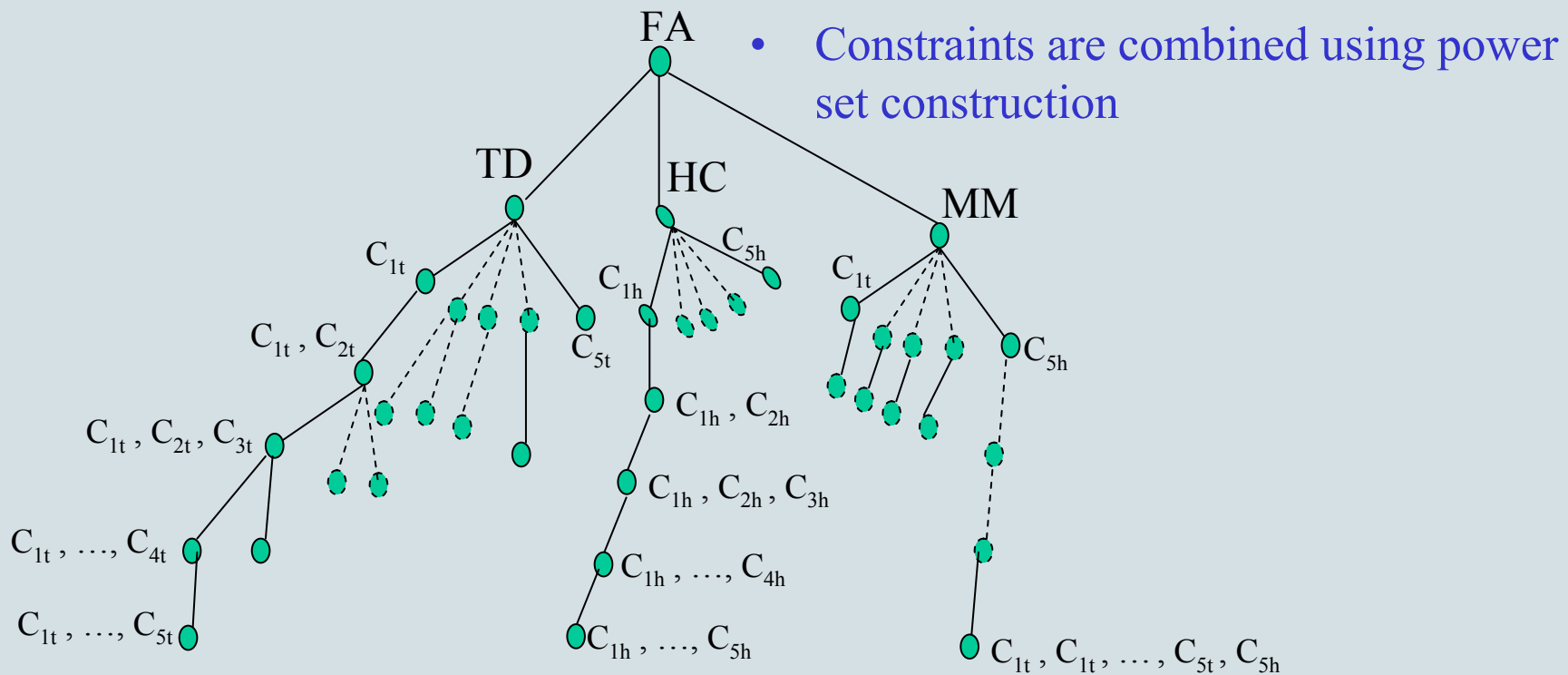
	$\Delta_t$	$\Delta_h$	$C_{1t}$	$C_{1h}$	$C_{2t}$	$C_{2h}$	$C_{3t}$	$C_{3h}$	$C_{4t}$	$C_{4h}$	$C_{5t}$	$C_{5h}$
TD	$\Delta_t$	$\phi$	F	F	F	F	F	F	0	0	0	0
HC	$\phi$	$\Delta_h$	F	F	F	F	F	F	0	0	0	0
SR:TD	$\Delta_t$	$\phi$	T	F	F	F	F	F	0	0	0	0
SR:HC	$\phi$	$\Delta_h$	F	T	F	F	F	F	0	0	0	0
MM	$\Delta_t$	$\Delta_h$	F	F	F	F	F	F	0	0	0	0
HC:JIT	$\phi$	$\Delta_h$	F	F	F	F	F	F	0	0	0	$ Q_h $
TD:JIT	$\Delta_t$	$\phi$	F	F	F	F	F	F	0	0	$ Q_t $	0
HC:LJIT	$\phi$	$\Delta_h$	F	F	F	F	F	F	0	m	0	m
TD:LJIT	$\Delta_t$	$\phi$	F	F	F	F	F	F	m	0	m	0

## A Simple Taxonomy Tree

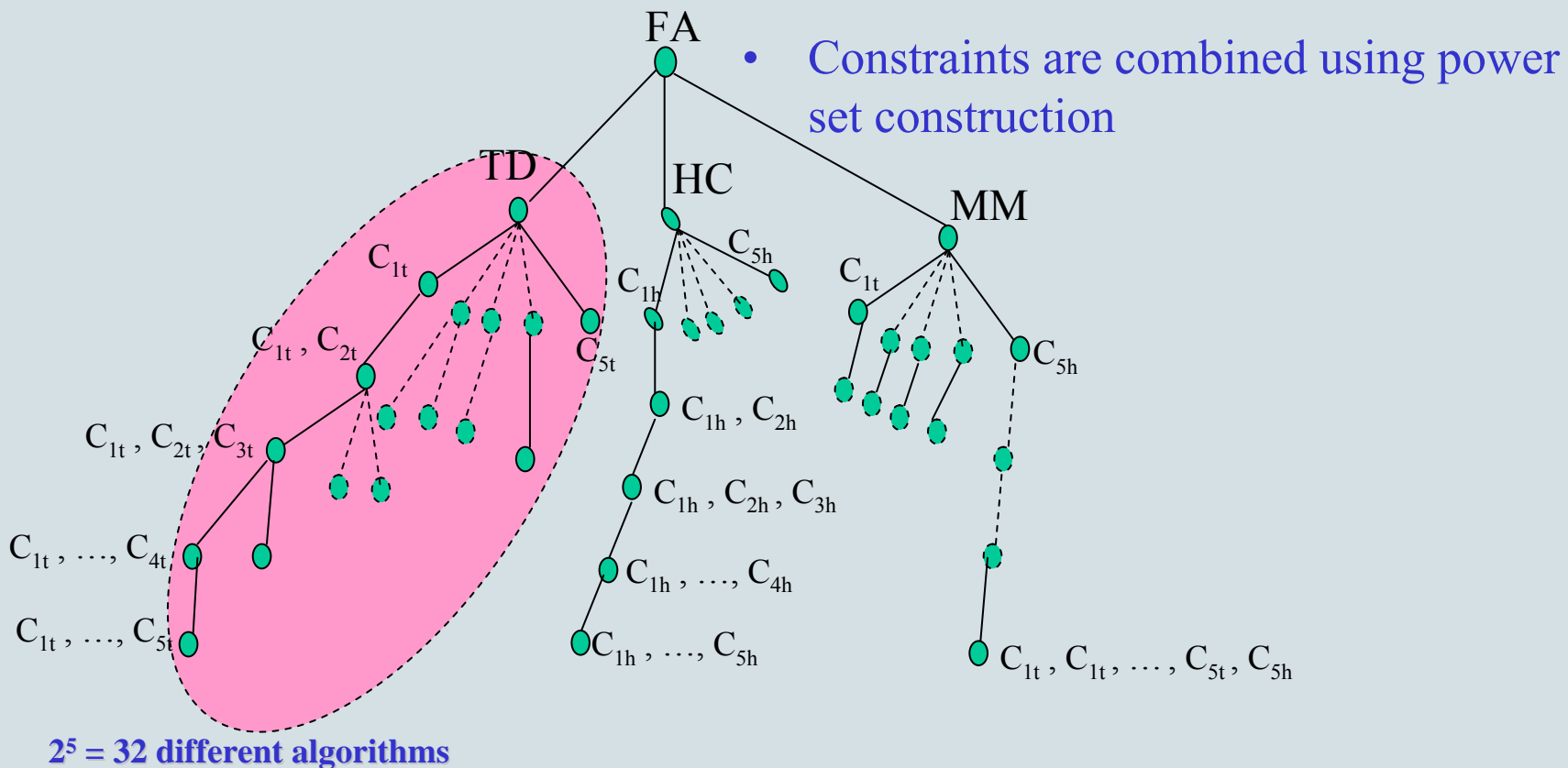
### Corresponding Toolkit Structure



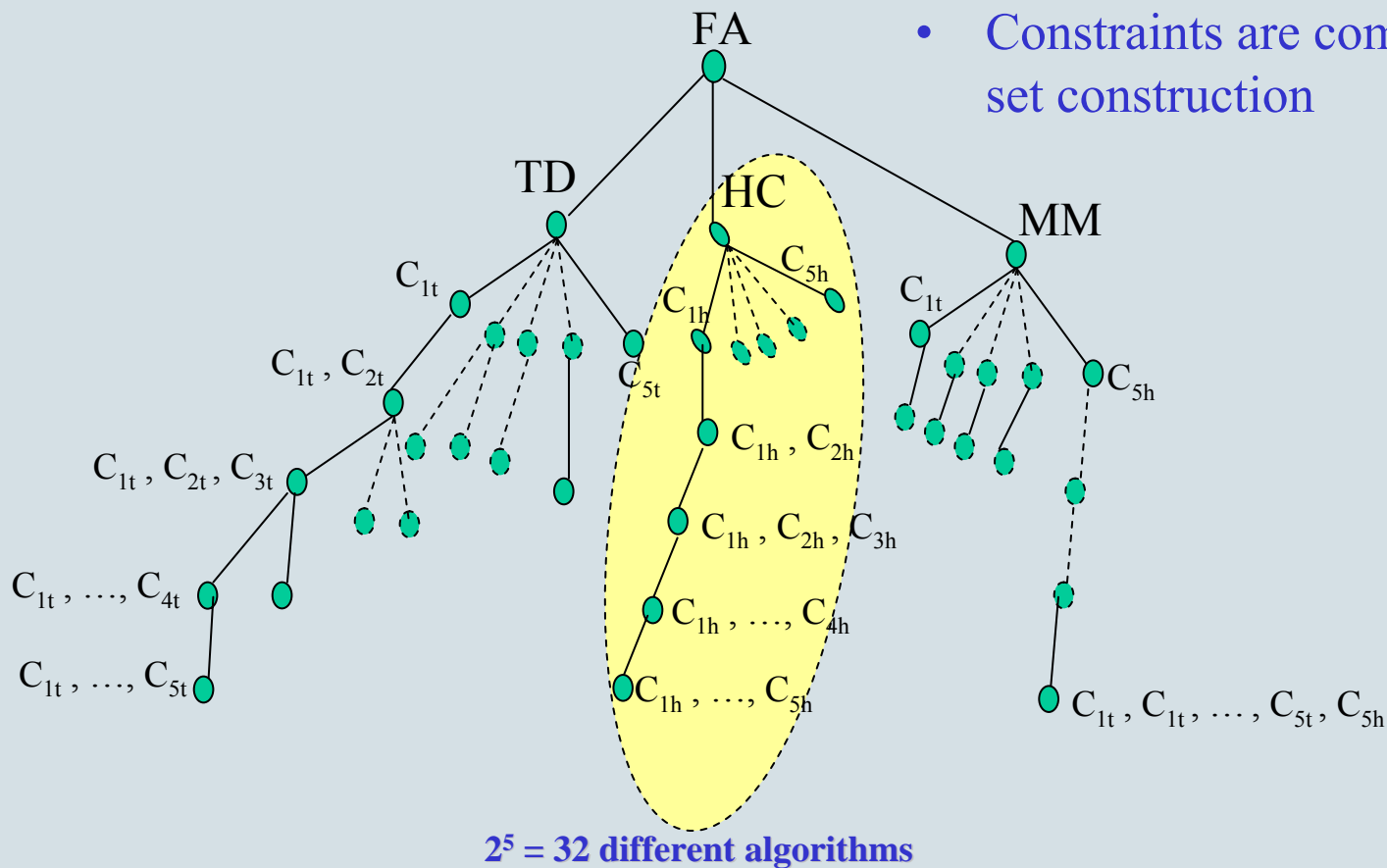
## A Preliminary Taxonomy Tree



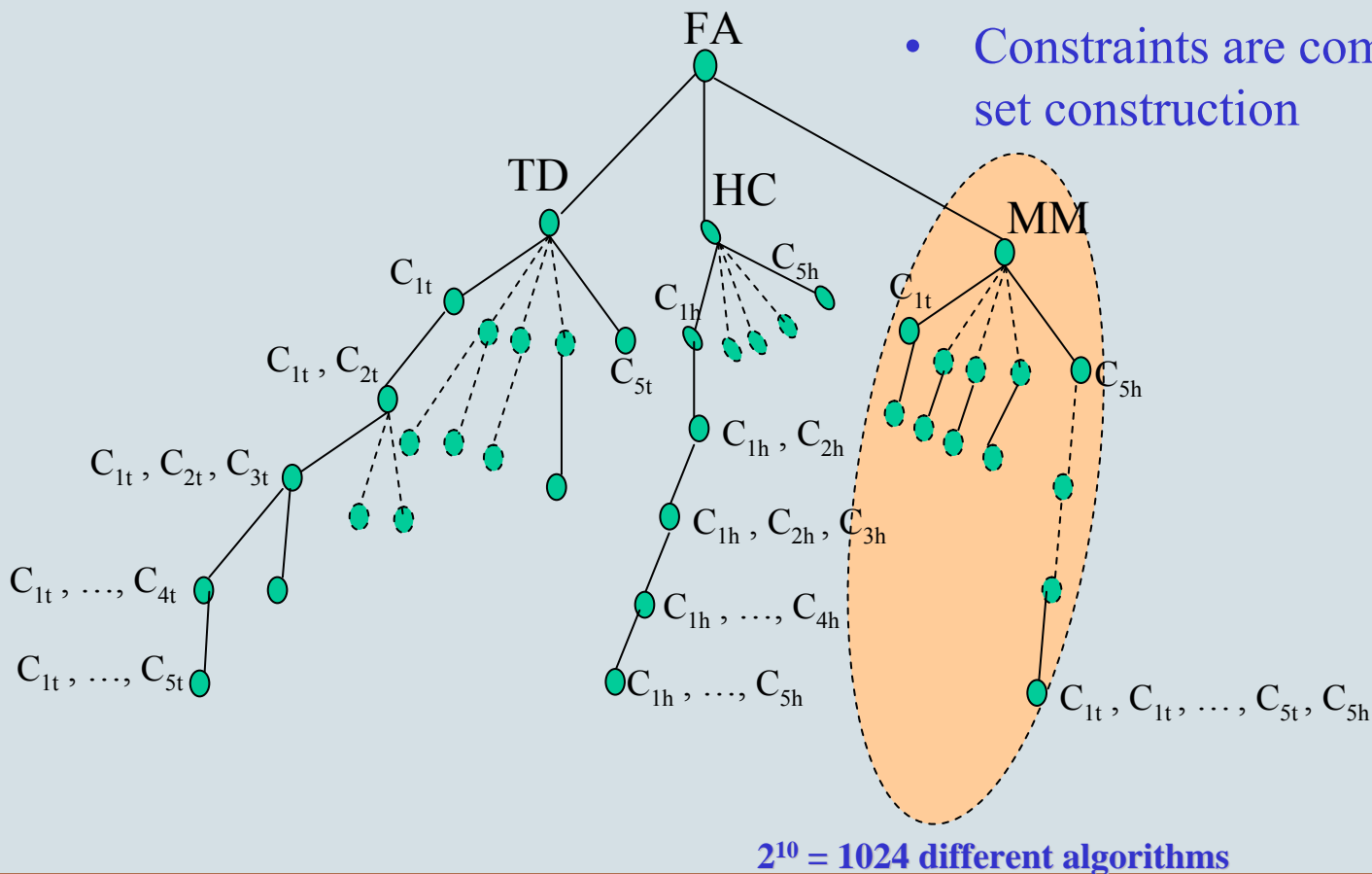
## A Preliminary Taxonomy Tree



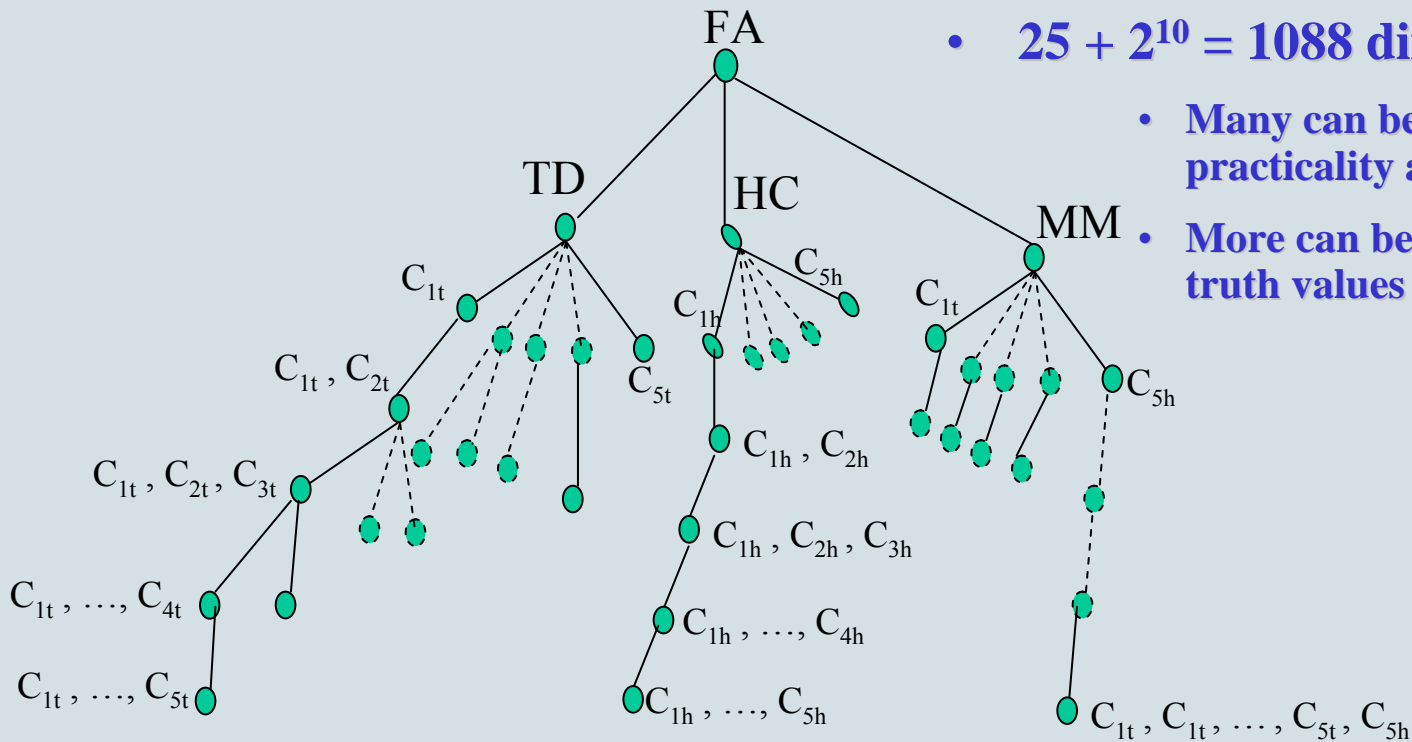
## A Preliminary Taxonomy Tree



## A Preliminary Taxonomy Tree



## A Preliminary Taxonomy Tree



- $25 + 2^{10} = 1088$  different algorithms
- Many can be removed according to practicality and redundancy
- More can be added by assigning truth values to boolean constraints

## Future Work

- Performance Analysis of the kwk Algorithms
  - Cache miss correlation
    - Effects of cache structure
    - Effects of cache capacity
    - Effects of mapping/replacement algorithms
  - Effects of pipelining
  - Effects of branch prediction
- Hardware Design Recommendations
- Applications of kwk algorithms
  - DNA, NIDS, Virus Scanning...
- Hardware Implementations
- Toolkit & DSL for FIRE Station

# Characterization of FAs Implementations

QUESTIONS?

# In Search of a Suitable logo for FASTAR