Recognizing regular tree languages with static information

Alain Frisch (ENS Paris)
Efficient compilation of patterns in XDuce/CDuce/…

E.g.:

```plaintext
type A = <a>[ A* ]
type B = <b>[ B* ]
let f ((A|B) -> Int) A -> 0 | B -> 1
let g ((A|B) -> Int) <a>_ -> 0 | _ -> 1
```

Encourage more declarative style in patterns.
The separation problem

- \( D \): domain of values
  
  Given \( X_1, \ldots, X_n \subset D \) (pairwise disjoint) and \( v \in \bigcup X_i \), returns \( i \) s.t. \( v \in X_i \)

- Two-stage process: the \( X_i \) come first (compile-time), and \( v \) comes after (run-time)

  Given \( X_1, \ldots, X_n \), computes a function \( s : \bigcup X_i \rightarrow \{1, \ldots, n\} \) s.t. \( \forall v \in \bigcup X_i, v \in X_{s(v)} \)

- We want \( s \) to be efficient, even if computing it is expensive.

- Parameters: the domain \( D \), the class of possible \( X_i \).

- Note: \( \bigcup X_i \) is the static information provided by the type system.
Examples

- $D$: integers; $X_i$: unions of intervals.
  - Decision tree (dichotomy)

- $D$: strings; $X_i$: finite sets.
  - Trie.
  - Can ignore suffixes.
  - E.g.: $X_1 = \{aa, ab, ac\}$, $X_2 = \{ba, bb, bc\}$

- $D$: strings; $X_i$: regular languages.
  - Deterministic automaton.

- $D$: XML documents; $X_i$: XML types (schema) = regular tree languages.
  - ???
Expected properties

- Traverse each node of the tree only a finite bounded number of times.
- Ignore subtrees that can be ignored
  - because of the static information and/or
  - because they are irrelevant.
- Independence w.r.t. the syntax of types (only the denoted sets matter).
Classical solutions

- **Backtracking tree automata**
  - Don’t guarantee linear time recognition.

- **Bottom-up deterministic tree automata.**
  - Only consider downward context (subtree).
  - Ignore upward context (path).
  - Cannot adapt their behavior to the current location in the tree.

- **Top-down deterministic tree automata**
  - Cannot recognize arbitrary regular tree languages.
  - Only consider the upward context.
  - No left-to-right propagation of information.
Contexts

▷ Path
▷ Subtree
▷ Left
Trees:

\[ v ::= a \mid (v_1, v_2) \]

(ie: \( V = \Sigma + V \times V \))

Regular languages defined by equations:

\[
\begin{align*}
X_1 &= \Sigma_1 \cup X_\ldots \times X_\ldots \cup \ldots \cup X_\ldots \times X_\ldots \\
\ldots \\
X_n &= \Sigma_n \cup X_\ldots \times X_\ldots \cup \ldots \cup X_\ldots \times X_\ldots
\end{align*}
\]

E.g.:

\[
\begin{align*}
X_0 &= X_1 \times X_2 \\
X_1 &= \{a\} \cup X_1 \times X_1 \\
X_2 &= \{b\} \cup X_2 \times X_2
\end{align*}
\]
Non-uniform automata

- A specific kind of push-down automata.
  - Combine the advantages of bottom-up and top-down TA.
  - Can totally ignore whole subtrees.
- Fixed traversal order.
- Thread a control-state through the traversal.
  - Accumulates knowledge gained from the traversal.
  - Depends on the left context (not only path).
Non-uniform automata
Non-uniform automata

Input: state $q$
Non-uniform automata

Input: state $q$
$q \mapsto q_1$
Non-uniform automata

Input: state $q$

$q \mapsto q_1$

$r_1 \in R(q_1)$
Non-uniform automata

Input: state $q$

$q \mapsto q_1$

$r_1 \in R(q_1)$

$q, r_1 \mapsto q_2$
Non-uniform automata

Input: state $q$

$q \mapsto q_1$

$r_1 \in R(q_1)$

$q, r_1 \mapsto q_2$

$r_2 \in R(q_2)$
Input: state $q$

$q \mapsto q_1$

$r_1 \in R(q_1)$

$q, r_1 \mapsto q_2$

$r_2 \in R(q_2)$

$q, r_1, r_2 \mapsto r \in R(q)$
Main technical contribution: a compilation algorithm that generates efficient NUA (which ignore many subtrees).

Key idea: propagate static information precisely (in the control state)
May ignore right subtree, according to left subtree.
Propagate static information.
Ignore right subtree.
Ignore left subtree (normalization).
Ignore right subtree.
Could also ignore left subtree.
Consider left subtree, but could ignore it.
Compilation of PM in CDuce

- Handle capture variables.
- All CDuce types (open/closed records, integer/characters intervals, ...).
- Effectively encourages declarative style without degrading performances.
Thank you!