Tree Transducers and Tree Adjoining Grammars

Historical and Current Perspectives

William C. Rounds
University of Michigan, Ann Arbor
Outline

• Some history – genesis of tree transducers and tree grammars
• A little bit on the genesis of feature logic
• A preliminary attempt to unify transducers, TAGs, and feature logic
• A few questions about ongoing work
The 60’s

A new religion is born...
The Master
I become a disciple
The Peters-Ritchie result

Transformational grammars are Turing-powerful

End times for TG?
TG survives!

It begins series of mutations into GB, PP, MIN, REC
A failed Math PhD?
Go into computer science, but call it math
Problem

• Go beyond context-free (reduplication phenomena)
• Have recursion
• Avoid Turing-powerful
• Have a vague resemblance to transformational grammar
Tree automata – salvation!
Thatcher, Wright, Brainerd, Doner, Rabin

- Tree automaton - recognizable = context-free.
- Top-down infinite tree automata emptiness is decidable
- Idea - use top-down to define tree transductions
- Reinforced by being able to model syntax-directed translation (not for NL!)
Top-down tree transduction
Further developments

- Tree transducers which could delay (create their own input), now called macro tree transducers
- One-state macro tree transducer = CFG on trees
- Santa Cruz 1970 – Thatcher, Joshi, Peters, Petrick, Partee, Bach: Tree Mappings in Linguistics
- Birth of TAGs, aka linear CFGs on trees
- Natural generalization to graph grammars, links to term rewriting
- Burgeoning industry in Europe, led by Engelfriet
What was I doing? (1976-1983)

• Some results on complexity
• Modelling semantics of concurrency
• Ignoring tree transducers
• Learning about bisimulations and modal logic
Learn from your graduate students

- Bob Kasper (1983-4): What is a disjunctive feature structure?
- How should these be unified?
- Write down desired laws for distributing unification over disjunction
- With the background of modal logics for concurrency, realize that feature structures are models for feature logic.
- PATR-2 actually invents feature logic; we extend to modal version.
- Make big mistakes proving a completeness theorem.
- Drew Moshier (1986-7), Larry Moss, Bob Carpenter fix things
Skip to the near present

- Probabilities, statistics, and corpora
- Resurgence of (weighted) finite-state transducers on strings as unifying model for speech recognition and generation algorithms (Mohri, Pereira, Riley)
- Kevin Knight and students propose probabilistic tree transducers as schemas for MT algorithms
- Multiplicity of tree transducer models (e.g., semilinear non-deleting deterministic, with inherited attributes and right-regular lookahead)
- Can we take any of these off the shelf and actually use them?
Two directions

- Use linguistic evidence to select relevant class of models (this workshop)
- Use various mathematical means to understand commonalities and differences among models
- Shieber: synchronous TAGS and tree transducers
- Rogers: TAGS as 3D tree automata
- Take a break from inventing the next variation
Model-theoretic syntax

• Long tradition of regarding generation as proof, even parsing as proof.

• In last ten years: what is the model theory for these proof systems?

• Best-known example: Montague grammar (focus on interpretation).

• Now: type-logical syntax (Morrill, Moortgat) and type-logical semantics (Carpenter).

• Feature logic is another description language for syntax.

• Attempts to view grammatical derivations as proofs, usually in logic programs with feature logic as a constraint language.

• HPSG: fully developed linguistic theory grounded in feature descriptions and unification; grammars as logical constraints on feature structures.
Clean proof theory and accompanying model theory for feature logic?

- Incomplete and ongoing work
- Goals: self-contained proof theory (do not glue onto grammar)
- Logic should model common grammatical formalisms, to understand them better
- Some previous work: Keller (extending feature logic to model TAGs); Vijay-Shanker and Joshi (FTAGs)
Three-dimensional trees (Rogers)
Adjunction as a 3D tree
Adjunction as a 3D FS

Initial tree

Auxiliary tree

Substitution
Example of Adjunction Rule

\[ n[c: \bot] \rightarrow \{3:n[l: adj[c: pretty], r: n[c: \bot]] \sqcup (c \equiv 3rc), 3:n[c: \bot] \sqcup (c \equiv 3c) \} \]
Rules as logical constraints

\[
\begin{align*}
n[c : \bot] & \rightarrow \{3 : n[l : \text{adj}[c : \text{pretty}], r : n[c : \bot]] \sqcup (c \doteq 3rc), \\
& \quad 3 : n[c : \bot] \sqcup (c \doteq 3c)\}
\end{align*}
\]
Quick look at tree transductions

\[ q[in : c[l : \perp, r : \perp]] \rightarrow 3 : c[l : p, r : q] \sqcup [inr = 3lin] \sqcup [inl = 3rin] \]
Quick look at tree transductions

\[ q[\text{in}: c[l:\bot, r:\bot}] \rightarrow 3: c[l:p, r:q] \sqcup [\text{inr} \doteq 3\text{lin}] \sqcup [\text{inl} \doteq 3\text{rin}] \]

\[ r \, q[\text{in}: c[l:\bot, r:\bot}] \rightarrow 3: c[l:p, r:q] \sqcup [\text{inr} \doteq 3\text{rin}] \sqcup [\text{inl} \doteq 3\text{lin}] \]
Quick look at tree transductions

\[ q[\text{in} : c[l : \bot, r : \bot]] \rightarrow 3 : c[l : p, r : q] \sqcup [\text{in}r \doteq 3\text{in}] \sqcup [\text{in}l \doteq 3\text{lin}] \]

\[ p[\text{in} : b] \rightarrow 3 : a \]
Theory behind this

• Disjunctive feature logic programming.

• A program is set of rules of the form $f \rightarrow L$, where $f$ is a feature structure, and $L$ is a clause, a finite set of feature structures.

• These rules can be used in proofs, to create a theory, in general an infinite set of clauses.

• A feature structure $m$ satisfies the clause $L$ if some element of $L$ subsumes it.

• $m$ is a model of the program if for any rule $f \rightarrow L$, if $f$ subsumes $m$, then $m$ satisfies $L$.

• Theorem: the minimal models of the program are the minimal structures satisfying all clauses of the theory.

• This way we can get infinite FS as models.

• There is a sound and complete resolution proof system to go with all of this.
The resolution rules

• Logical resolution:

\[
\begin{align*}
K & \quad L \\
\text{f} & \in K \\
\text{g} & \in L \\
\text{f} \cup \text{g} & \models M \\
\hline
M \cup (K \setminus \{f\}) \cup (L \setminus \{g\})
\end{align*}
\]

where \(K, L, M\) are clauses.

• Clause introduction (nonlogical resolution):

\[
\begin{align*}
M & \quad \text{g} \in M \\
\text{f} \rightarrow \text{L} & \in \text{P} \\
\text{f} \subseteq \text{g} \\
\hline
\text{L} \cup (M \setminus \{g\})
\end{align*}
\]
Questions and further work

• Can you compile FL specifications into a parser?
• What about other formalisms, like synchronous TAGS?
• Can you work probabilities, or more generally, weights, into a fully declarative formalism?
Thanks!