Kahina, a Debugging Framework for Logic Programs and TRALE

Johannes Dellert, Kilian Evang, Frank Richter
Seminar für Sprachwissenschaft
Eberhard Karls Universität Tübingen

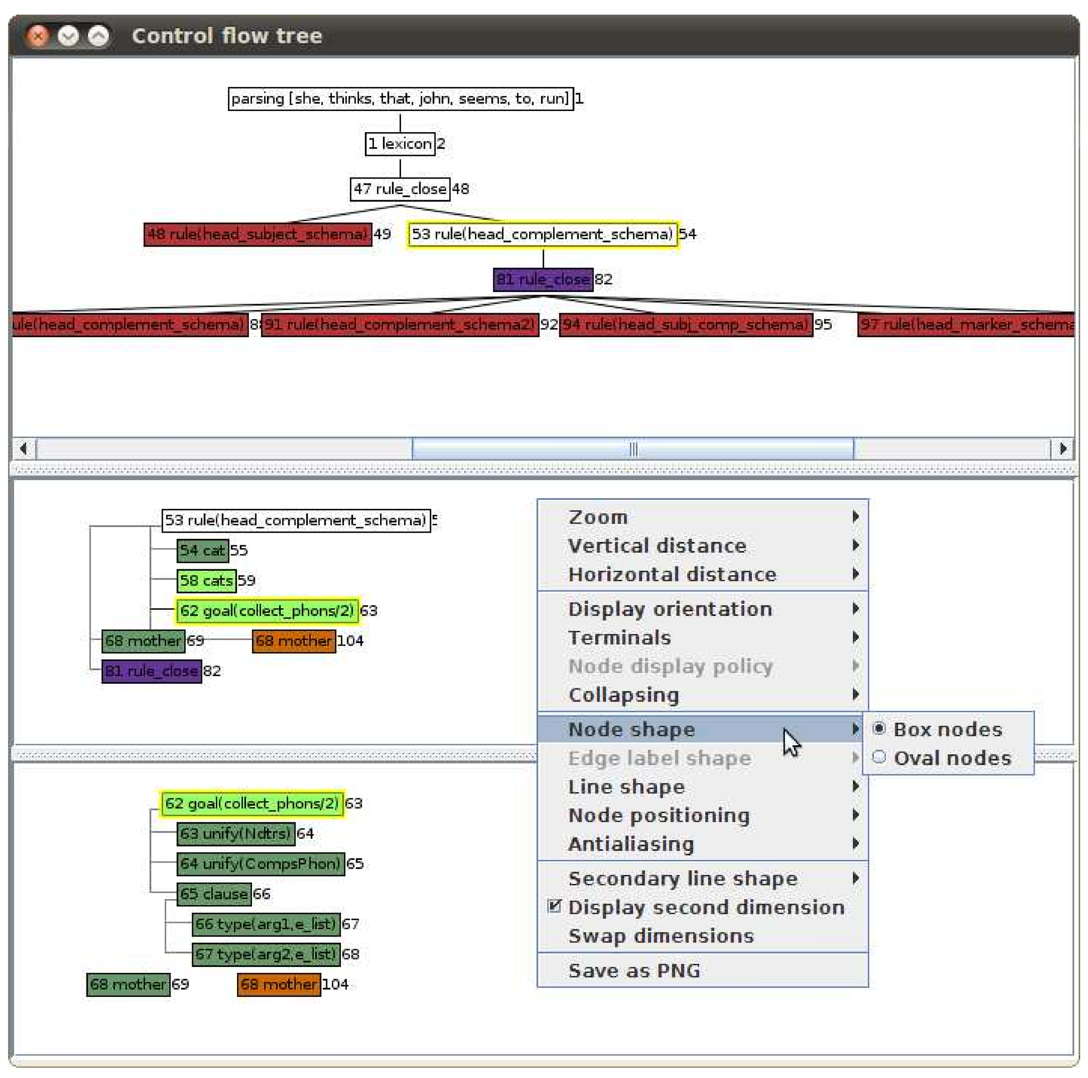
Debugging of Grammars for Complex Parsing Systems

- Debugging large-scale TRALE grammars that stay true to their declarative specification in HPSG is a particularly challenging logic programming task, due to the memory burden involved in keeping track of huge parsing processes, including intricate interactions between rules, constraint applications, and suspended goals in co-routining.
- These challenges can make debugging grammars very time-consuming, and, in certain situations and without appropriate high-level tools, practically impossible.
- The situation is similar for debugging grammars using other popular formalisms such as TAG or LFG, as well as for larger logic programs in general. The few existing graphical tools for Prolog debugging are tailored to smaller problems and do not scale well. Closing this technological gap presupposes the development of innovative concepts and tools.

Fine-grained Control over Parsing Processes

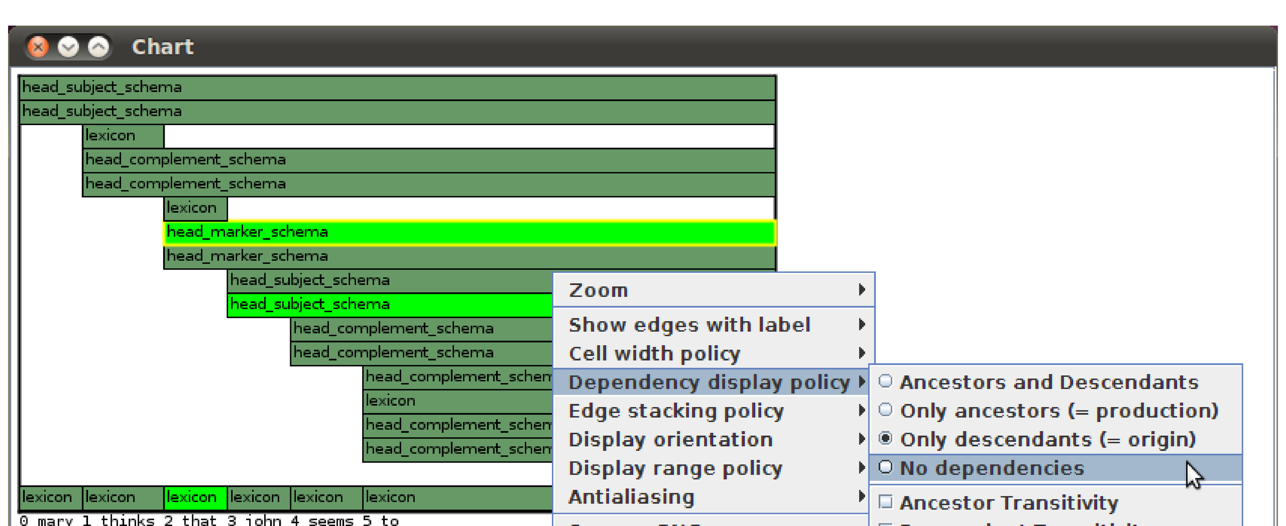
- The Kahina debugging environment gives the user fine-grained control over complex parsing processes. For TRALE debugging, a Kahina-based graphical interface is used to control a TRALE instance using the commands of a classical Prolog tracer.
- Unlike a classical Prolog tracer, Kahina logs the relevant data for each parsing step and stores them in full detail for post-mortem analysis. These data can then be inspected and analyzed using a variety of visualization modules. All these modules are tightly integrated to allow the user to view the relevant data from different angles, discover new connections and detect interesting patterns.

Integrated Control Flow and Decision Tree



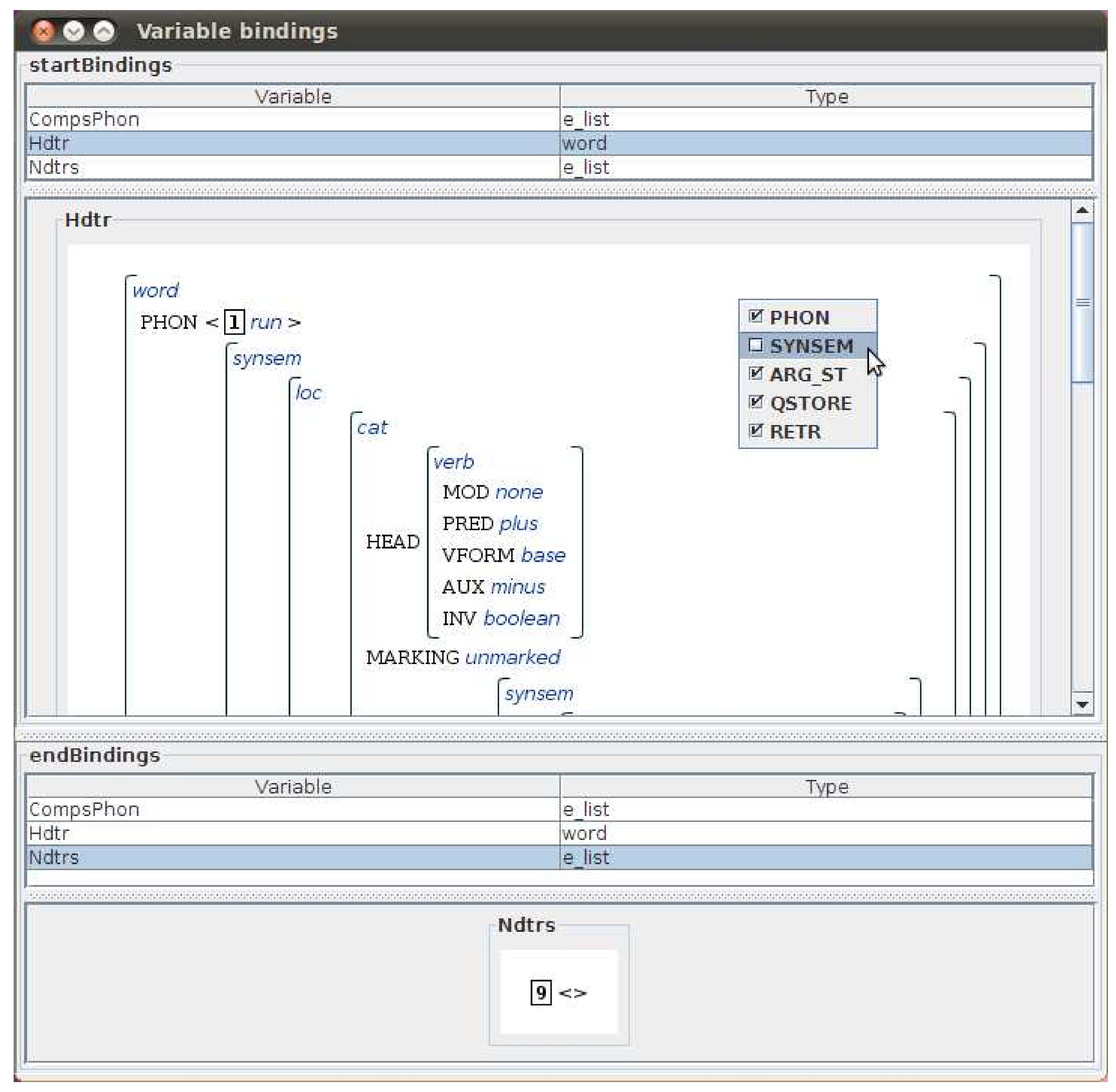
- The details for all steps are accessible via selection of nodes in a **step tree**.
- The tree view is partitioned into three configurable layers for navigation with different levels of detail. This decomposes the possibly huge tree into meaningful navigable units, based on the notion of **tree layers** that rank nodes by importance.
- Two dimensions in a single view allow instant access to all relevant information about the WHEN and WHY of a step: the **search dimension** (primary tree structure) shows the effects of backtracking, whereas the **call dimension** (secondary tree structure: indentation and lines) reflects program structure.
- There are numerous aids for browsing complex structures with huge numbers of nodes: coloring functions for nodes depending on their properties, subtree collapsing for pruning the visualization to suit current needs, and a history of inspected steps.

Interactive Chart Display



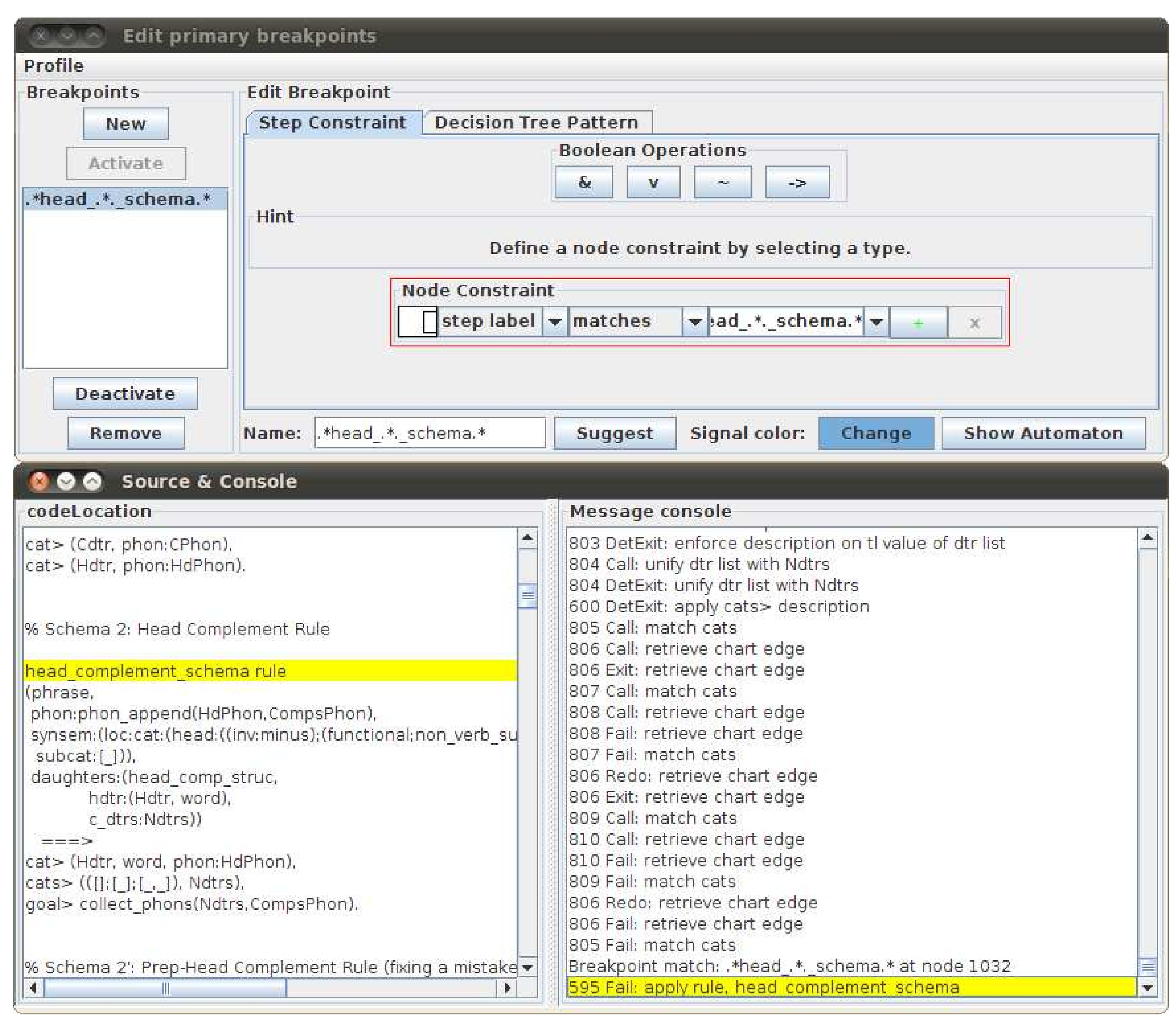
- The chart consists of **edges** representing established constituents and gives a very compact overview of the parsing process. When an edge is selected, highlights show which other edges took part in deriving it, and the step tree jumps to the derivation details.
- Optional display of **failed edges** (not in picture) provides the user with direct links to explanations why some rule or principle could not be applied in the expected way.

Inspection of Variable Bindings



- At each step, this component shows the variables occurring in the currently relevant rule, constraint (principle), or other description, together with their values.
- Since variables in TRALE are always bound to **feature structures**, Kahina integrates the feature structure viewer GraleJ, which is also used in another component for displaying **local trees** corresponding to the current rule application.
- The variable bindings display also enables very detailed tracing of feature structure unification guided by **highlights** on the parts being processed at each step.

Defining Breakpoints and Automatizing Tasks via Tree Automata



- Often, the user will want to detect problematic and interesting configurations while skipping through the parsing process. By using **breakpoints**, the user can study and understand parsing processes that would be impossible to trace by hand.
- Kahina's powerful breakpoint system is built around **pattern matches** in the step tree. A specialized on-line variant of bottom-up **tree automata** allows efficient pattern matching while the step tree is growing. Matches are shown in the **message console**. This console also contains system messages that explain every parse step.
- A **tree pattern editor** makes it possible to define interesting patterns in an intuitive way. Tree patterns can be grouped into profiles, activated/deactivated, and imported/exported for exchange.
- Kahina also employs tree automata for automatization purposes. Beyond on-line and off-line pattern matching, it also supports the definition of **skip points**, **creep points**, and **fail points** to steer the parsing process.

This project is supported by the MFG (Medien- und Filmgesellschaft Baden-Württemberg) through a Karl-Steinbuch-Stipendium. For references and further information visit us at **www.kahina.org**.