Compiler verification

Verified PEG Parser

From "TRX: A Formally Verified Parser Interpreter"
A. Koprowski and H. Binsztok, 2011

Neven Villani

2021-12-14

Motivation

- At time of writing, CompCert started at the AST: the parsing step is not verified
- But programming is not done directly with ASTs
- This is an attempt at filling this missing initial step

Choice of tools

- Parser extracted from Coq
- Grammar expressed as a PEG

Parsing Expression Grammar

Definitions, Advantages

Syntax

e	$::=$	ε	empty
	$\mid$	$[a]$	terminal
	$\mid$	A	nonterminal
	$\mid$	$e_1; e_2$	sequence
	$\mid$	e_1/e_2	prioritized choice
	$\mid$	e^*	greedy repetition
	$\mid$	$!e$	lookahead

Nonprimitives

$$
\begin{aligned}
[\text{``}a_1 \cdots a_n\text{''}] &\quad := \quad [a_1]; \cdots ; [a_n] \\
[a - z] &\quad := \quad [a] / \cdots / [z] \\
e^+ &\quad := \quad e; e^* \\
e? &\quad := \quad e / \varepsilon \\
\&e &\quad := \quad !!e
\end{aligned}
$$

Examples

$\{a^n b^n \mid n\}$

$$X_{a,b} := ([a]; X_{a,b}; [b])?$$

$$S_{a,b} := X_{a,b}; ![\cdot]$$

palindromes

$$Y := ([a]; Y; [a])/([b]; Y; [b])/a/b/\varepsilon$$

$$P := Y; ![\cdot]$$

$\{a^n b^n c^n \mid n\}$

$$S' := \&(X_{ab}; [c]^*); ([a]^*; X_{bc}); ![\cdot]$$

Semantics

Basic rules:

$$\frac{}{(\varepsilon, s) \rightsquigarrow \surd_{s}^{()}} \qquad \frac{}{([\cdot], []) \rightsquigarrow \bot} \qquad \frac{}{([x], x :: s) \rightsquigarrow \surd_{s}^{x}}$$

$$\frac{(e_1, s) \rightsquigarrow \bot}{(e_1; e_2, s) \rightsquigarrow \bot} \qquad \frac{(e, s) \rightsquigarrow \surd_{s'}^{v} \qquad (e', s') \rightsquigarrow \surd_{s''}^{v'}}{(e; e', s) \rightsquigarrow \surd_{s''}^{(v,v')}}$$

$$\frac{(e, s) \rightsquigarrow \bot}{(e^*, s) \rightsquigarrow \surd_{s}^{[]}} \qquad \frac{(e, s) \rightsquigarrow \surd_{s'}^{v} \qquad (e^*, s') \rightsquigarrow \surd_{s''}^{V}}{(e^*, s) \rightsquigarrow \surd_{s''}^{v::V}}$$

Semantics (cont)

Some more important rules:

$$\frac{(e, s) \rightsquigarrow \surd_s^v}{(!e, s) \rightsquigarrow \bot} \qquad \frac{(e, s) \rightsquigarrow \bot}{(!e, s) \rightsquigarrow \surd_s^{()}}$$

$$\frac{(e, s) \rightsquigarrow \surd_{s'}^v}{(e/e', s) \rightsquigarrow \surd_{s'}^v} \qquad \frac{(e, s) \rightsquigarrow \bot \qquad (e', s) \rightsquigarrow r}{(e/e', s) \rightsquigarrow m + n + 1r}$$

Advantages

- No separate lexing step
- No large parsing table
- Unambiguous, fully deterministic
- More expressive than context-free

Contribution

TRX is proven correct.
If $\mathcal{G}$ is syntactically valid then:

$$\begin{aligned}
&\texttt{well-formed}(\mathcal{G}) = \texttt{true} \implies \forall s. \\
&\qquad (\texttt{parse}(\mathcal{G}, s) \text{ terminates}) \\
&\qquad \wedge (\forall v, s'.\ \texttt{parse}(\mathcal{G}, s) = \texttt{Ok}(v, s') \iff \exists m.\ (v_0, s) \rightsquigarrow \sqrt{^{v}_{s'}}) \\
&\qquad \wedge (\texttt{parse}(\mathcal{G}, s) = \texttt{Err} \iff \exists m.\ (v_0, s) \rightsquigarrow \bot)
\end{aligned}$$

(slightly redundant)
Appropriate characterization since $\rightsquigarrow$ is fully deterministic

Limitations

- Exponential time (can be solved with memoization)

$$S := A; ![.]$$
$$A := ([a]; A; [b])/([a]; A; [c])/\varepsilon$$

$$a^n c^n$$

- Nontermination

$$S := A; ![.]$$
$$A := A?; [a]$$

Termination

Well-formedness, Motivation, Limitations

Termination

- Source of nontermination is (mutual) left recursion
- Not so easy to check

$$A := B/(C; !D; A)$$

  B can fail, C can succeed without consuming, D can fail

- Completeness is undecidable
- Well-formedness is easily checkable and implies completeness

Well-formedness

$e \in \mathbb{P}_0$ if e can accept without consuming input
WF fixpoint of well-formedness rules

$$\frac{}{\varepsilon \in \mathsf{WF}} \qquad \frac{}{[a] \in \mathsf{WF}} \qquad \frac{e \in \mathsf{WF}}{!e \in \mathsf{WF}}$$

$$\frac{e \in \mathsf{WF} \qquad e \in \mathbb{P}_0 \Rightarrow e' \in \mathsf{WF}}{e; e' \in \mathsf{WF}} \qquad \frac{e \in \mathsf{WF} \qquad e' \in \mathsf{WF}}{e/e' \in \mathsf{WF}}$$

$$\frac{e \in \mathsf{WF} \qquad e \notin \mathbb{P}_0}{e^* \in \mathsf{WF}}$$

Well-formedness implies termination

All subexpressions of the grammar

$$E(\mathcal{G}) = \left\{ e' \mid e' \sqsubseteq e_A, A \text{ nonterminal} \right\}$$

$\mathcal{G}$ is well-formed if $E(\mathcal{G}) \subseteq \mathsf{WF}$

This is an overapproximation

$$A := !\varepsilon; A \text{ terminates but is rejected}$$

Usability

Ergonomics, Performance, Improvements

Specification

Most parser interpreters provide a DSL to define the grammar.
Not the case of TRX.

- Grammar specified within Coq
- Correctness proof is easy
- Requires familiarity with Coq

Bootstrapping issues

- Parsing grammar description would require a type that contains itself
- Another solution: code generation (significantly more difficult)

Performance

tool	XML parser	Java parser
JAXP	2.3s.	
JavaCC		23.0s.
TRX-gen	5.1s.	25.5s.
TRX-int	40.0s.	289.3s.
TRX-cert	128.9s.	662.4s.
Mouse	206.4s.	269.6s.

Figure 8, page 20

Performance of TRX-cert on two examples

TRX-cert: certified version
TRX-int: prototype with similar functionality
TRX-gen: code generating parser

Performance diagnosis

- verification overhead – steps with only logical use
  - not significant
- extraction overhead – extracted code under-performs
  - conversions (probably optimized)
  - `List.rev` is quadratic
  - `Char` is a Coq natural
  - `Ascii` is 8 booleans
  - garbage collection spikes
- algorithmic overhead – suboptimal algorithm

Summary

Characterization

$$
\begin{aligned}
&\texttt{well-formed}(\mathcal{G}) = \texttt{true} \implies \forall s.\\
&\qquad (\texttt{parse}(\mathcal{G}, s) \text{ terminates})\\
&\qquad \land (\forall v, s'.\ \texttt{parse}(\mathcal{G}, s) = \texttt{Ok}(v, s') \iff \exists m.\ (v_0, s) \rightsquigarrow \surd^{v}_{s'})\\
&\qquad \land (\texttt{parse}(\mathcal{G}, s) = \texttt{Err} \iff \exists m.\ (v_0, s) \rightsquigarrow \bot)
\end{aligned}
$$

Performance

tool	XML parser	Java parser
JAXP	2.3s.	
JavaCC		23.0s.
TRX-gen	5.1s.	25.5s.
TRX-int	40.0s.	289.3s.
TRX-cert	128.9s.	662.4s.
Mouse	206.4s.	269.6s.