TRX: A Formally Verified Parser Interpreter

Adam Koprowski Henri Binsztok

MLstate, Paris, France
http://mlstate.com

European Symposium on Programming (ESOP '10)
22 March

Outline

Context

- SaaS (Software as a service) is a new model of software deployment that is gaining popularity.
- OPA (One-Pot Application) is a new unified technology for developing rich web applications.
  The language has built-in: persistency, parsing capabilities, client-code generation capabilities, concurrency, ...
- Safety is a distinguished feature of OPA:
  - OPA features a number of static checks for its applications,
  - OPA has a verification platform build on top of it [WIP],
  - OPA components will be subjected to formal verification [WIP].

MLstate

Context

- SaaS (Software as a service) is a new model of software deployment that is gaining popularity.

- OPA (One-Pot Application) is a new unified technology for developing rich web applications.
  The language has built-in: persistency, parsing capabilities, client-code generation capabilities, concurrency, . . .
- Safety is a distinguished feature of OPA:
  - OPA features a number of static checks for its applications

MLstate

Context

- SaaS (Software as a service) is a new model of software deployment that is gaining popularity.
- OPA (One-Pot Application) is a new unified technology for developing rich web applications.
  The language has built-in: persistency, parsing capabilities, client-code generation capabilities, concurrency, ...
- Safety is a distinguished feature of OPA:
  - OPA features a number of static checks for its applications,
  - OPA has a verification platform build on top of it [WIP],
  - OPA components will be subjected to formal verification [WIP].

Context

- SaaS (Software as a service) is a new model of software deployment that is gaining popularity.
- OPA (One-Pot Application) is a new unified technology for developing rich web applications.
  The language has built-in: persistency, parsing capabilities, client-code generation capabilities, concurrency, ...

```
// Database

db /wiki: stringmap(string)
db /wiki = "this is a new page"
save(name,content) = /wiki[name] = jQuery.getVal(#(content)); []

// User interface
text(t) =
  <textarea rows={5} cols={80} id="text">{t}</textarea>

page(name) =
  <h1>{ name:string }</h1>
  <p>{ text(/wiki[name]) }</p>
  <a onclick={save(name, "text")}>save</a>

main(name) = html("MLstate wiki :: {name}", page(name))

/* Dispatching urls: http://www.myserver.com/some_note should
   point to note some_note */
urls = parser | "/" ([A-Za-z]+) -> main(Text.to_string( __2 ))
              | "/"             -> main("Home")

// Starting the application and setting a default port
server = simple_server(urls, 2009)
```

- Safety is a distinguished feature of OPA:
  - OPA features a number of static checks for its applications,
  - OPA has a verification platform build on top of it [WIP],
  - OPA components will be subjected to formal verification [WIP].

Context

- SaaS (Software as a service) is a new model of software deployment that is gaining popularity.
- OPA (One-Pot Application) is a new unified technology for developing rich web applications.
  The language has built-in: persistency, parsing capabilities, client-code generation capabilities, concurrency, ...
- Safety is a distinguished feature of OPA:
  - OPA features a number of static checks for its applications,
  - OPA has a verification platform build on top of it [WIP],
  - OPA components will be subjected to formal verification [WIP].

MLstate

Context

- SaaS (Software as a service) is a new model of software deployment that is gaining popularity.
- OPA (One-Pot Application) is a new unified technology for developing rich web applications.
  The language has built-in: persistency, parsing capabilities, client-code generation capabilities, concurrency, ...
- Safety is a distinguished feature of OPA:
  - OPA features a number of static checks for its applications,
  - OPA has a verification platform build on top of it [WIP],
  - OPA components will be subjected to formal verification [WIP].

Context

- SaaS (Software as a service) is a new model of software deployment that is gaining popularity.
- OPA (One-Pot Application) is a new unified technology for developing rich web applications.
  The language has built-in: persistency, parsing capabilities, client-code generation capabilities, concurrency, ...
- Safety is a distinguished feature of OPA:
  - OPA features a number of static checks for its applications,
  - OPA has a verification platform build on top of it [WIP],
  - OPA components will be subjected to formal verification [WIP].

MLstate

Context

- SaaS (Software as a service) is a new model of software deployment that is gaining popularity.
- OPA (One-Pot Application) is a new unified technology for developing rich web applications.
  The language has built-in: persistency, parsing capabilities, client-code generation capabilities, concurrency, ...
- Safety is a distinguished feature of OPA:
  - OPA features a number of static checks for its applications,
  - OPA has a verification platform build on top of it [WIP],
  - OPA components will be subjected to formal verification [WIP].

Context

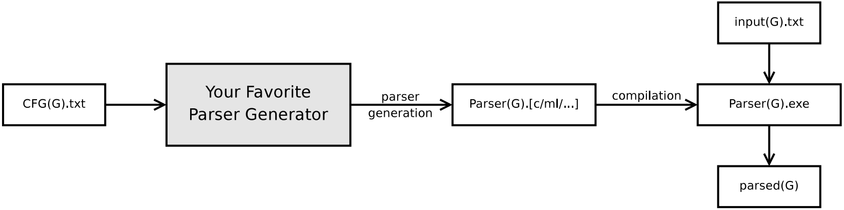
- SaaS (Software as a service) is a new model of software deployment that is gaining popularity.
- OPA (One-Pot Application) is a new unified technology for developing rich web applications.
  The language has built-in: persistency, parsing capabilities, client-code generation capabilities, concurrency, ...
- Safety is a distinguished feature of OPA:
  - OPA features a number of static checks for its applications,
  - OPA has a verification platform build on top of it [WIP],
  - OPA components will be subjected to formal verification [WIP].

Objective

- Parsing is a well-known and well-studied topic.
- Typical approach to parsing is by means of parser generators:
- … can we use formal mehods to verify some properties of generated parsers?

Objective

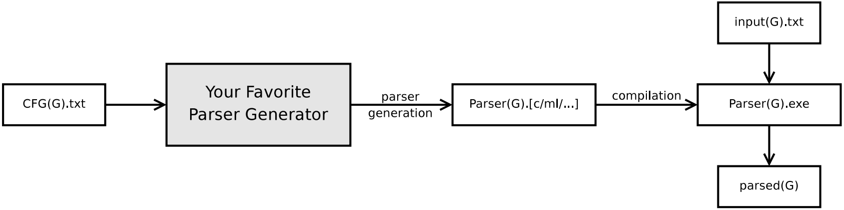
- Parsing is a well-known and well-studied topic.
- Typical approach to parsing is by means of parser generators:



- ... can we use formal mehods to verify some properties of generated parsers?

Objective

- Parsing is a well-known and well-studied topic.
- Typical approach to parsing is by means of parser generators:



- ... can we use formal mehods to verify some properties of generated parsers?

Parsing expressions

Main reference

Bryan Ford
Parsing expression grammars: a recognition-based syntactic foundation
POPL '04

Parsing expressions

Definition

Parsing expressions over non-terminals $\mathcal{V}_N$ and terminals $\mathcal{V}_T$:

$$
\begin{array}{rlrl}
\Delta := & \epsilon & \text{empty expression} & \\
| & [\cdot] & \text{any character} & \\
| & [a] & \text{a terminal symbol} & (a \in \mathcal{V}_T) \\
| & [\text{“}s\text{”}] & \text{a literal} & (s \in \mathcal{S}) \\
| & [a\text{–}z] & \text{a range} & (a, z \in \mathcal{V}_T) \\
| & A & \text{a non-terminal} & (A \in \mathcal{V}_N) \\
| & e_1 ; e_2 & \text{a sequence} & (e_1, e_2 \in \Delta) \\
| & e_1 / e_2 & \text{a } \textit{prioritized} \text{ choice} & (e_1, e_2 \in \Delta) \\
| & e* & \text{a zero-or-more } \textit{greedy} \text{ repetition} & (e \in \Delta) \\
| & e+ & \text{a one-or-more } \textit{greedy} \text{ repetition} & (e \in \Delta) \\
| & e? & \text{an optional expression} & (e \in \Delta) \\
| & !e & \text{a not-predicate} & (e \in \Delta) \\
| & \&e & \text{an and-predicate} & (e \in \Delta)
\end{array}
$$

Parsing expressions

Definition

Parsing expressions over non-terminals $\mathcal{V}_N$ and terminals $\mathcal{V}_T$:

$$
\begin{array}{rlrl}
\Delta := & \epsilon & \text{empty expression} & \\
| & [\cdot] & \text{any character} & \\
| & [a] & \text{a terminal symbol} & (a \in \mathcal{V}_T) \\
| & [\text{“}s\text{”}] := [s_1]\,;\,\ldots\,;\,[s_n] & \text{a literal} & (s \in \mathcal{S}) \\
| & [a\text{–}z] := [a] \;\;/\;\; \ldots \;\;/\;\; [z] & \text{a range} & (a, z \in \mathcal{V}_T) \\
| & A & \text{a non-terminal} & (A \in \mathcal{V}_N) \\
| & e_1; e_2 & \text{a sequence} & (e_1, e_2 \in \Delta) \\
| & e_1/e_2 & \text{a \textit{prioritized} choice} & (e_1, e_2 \in \Delta) \\
| & e* & \text{a zero-or-more \textit{greedy} repetition} & (e \in \Delta) \\
| & e+ := e; e* & \text{a one-or-more \textit{greedy} repetition} & (e \in \Delta) \\
| & e? := e/\epsilon & \text{an optional expression} & (e \in \Delta) \\
| & !e & \text{a not-predicate} & (e \in \Delta) \\
| & \&e := !!e & \text{an and-predicate} & (e \in \Delta)
\end{array}
$$

Parsing expressions

Definition

Parsing expressions over non-terminals $\mathcal{V}_N$ and terminals $\mathcal{V}_T$:

$$
\begin{array}{rlrl}
\Delta := & \epsilon & \text{empty expression} & \\
| & [\cdot] & \text{any character} & \\
| & [a] & \text{a terminal symbol} & (a \in \mathcal{V}_T) \\
| & [\text{“}s\text{”}] := [s_1] \,;\, \ldots \,;\, [s_n] & \text{a literal} & (s \in \mathcal{S}) \\
| & [a\text{–}z] := [a] \;/\; \ldots \;/\; [z] & \text{a range} & (a, z \in \mathcal{V}_T) \\
| & A & \text{a non-terminal} & (A \in \mathcal{V}_N) \\
| & e_1 ; e_2 & \text{a sequence} & (e_1, e_2 \in \Delta) \\
| & e_1 / e_2 & \text{a \textit{prioritized} choice} & (e_1, e_2 \in \Delta) \\
| & e* & \text{a zero-or-more \textit{greedy} repetition} & (e \in \Delta) \\
| & e+ := e ; e* & \text{a one-or-more \textit{greedy} repetition} & (e \in \Delta) \\
| & e? := e/\epsilon & \text{an optional expression} & (e \in \Delta) \\
| & !e & \text{a not-predicate} & (e \in \Delta) \\
| & \&e := !!e & \text{an and-predicate} & (e \in \Delta)
\end{array}
$$

PEG example

Example

A very simple PEG for mathematical expressions:

$$
\begin{array}{rcl}
\mathtt{ws} & := & ([\textvisiblespace] \;/\; [\backslash t])* \\
\mathtt{number} & := & [0\text{–}9]+ \\
\mathtt{term} & := & \mathtt{ws}\ \mathtt{number}\ \mathtt{ws} \;/\; \mathtt{ws}\ [(]\ \mathtt{expr}\ [)]\ \mathtt{ws} \\
\mathtt{factor} & := & \mathtt{term}\ [*]\ \mathtt{factor} \;/\; \mathtt{term} \\
\mathtt{expr} & := & \mathtt{factor}\ [+]\ \mathtt{expr} \;/\; \mathtt{factor}
\end{array}
$$

Example

How about making addition left-associative?

$$\mathtt{expr} := \mathtt{expr}\ [+]\ \mathtt{factor} \;/\; \mathtt{factor}$$

... but that makes the grammar *left-recursive*, leading to a *non-terminating* parser.

PEG example

Example

A very simple PEG for mathematical expressions:

$$
\begin{aligned}
\texttt{ws} &:= ([\textvisiblespace] \;/\; [\backslash t])* \\
\texttt{number} &:= [0\text{–}9]+ \\
\texttt{term} &:= \texttt{ws number ws} \;/\; \texttt{ws} \; [(] \; \texttt{expr} \; [)] \; \texttt{ws} \\
\texttt{factor} &:= \texttt{term} \; [*] \; \texttt{factor} \;/\; \texttt{term} \\
\texttt{expr} &:= \texttt{factor} \; [+] \; \texttt{expr} \;/\; \texttt{factor}
\end{aligned}
$$

Example

How about making addition left-associative?

$$\texttt{expr} := \texttt{expr} \; [+] \; \texttt{factor} \;/\; \texttt{factor}$$

... but that makes the grammar *left-recursive*, leading to a *non-terminating* parser.

PEG example

Example

A very simple PEG for mathematical expressions:

$$
\begin{aligned}
\texttt{ws} &:= ([\textvisiblespace] \;/\; [\backslash t])* \\
\texttt{number} &:= [0\text{–}9]+ \\
\texttt{term} &:= \texttt{ws number ws} \;/\; \texttt{ws}\; [(]\; \texttt{expr}\; [)]\; \texttt{ws} \\
\texttt{factor} &:= \texttt{term}\; [*]\; \texttt{factor} \;/\; \texttt{term} \\
\texttt{expr} &:= \texttt{factor}\; [+]\; \texttt{expr} \;/\; \texttt{factor}
\end{aligned}
$$

Example

How about making addition left-associative?

$$\texttt{expr} := \texttt{expr}\; [+]\; \texttt{factor} \;/\; \texttt{factor}$$

... but that makes the grammar *left-recursive*, leading to a *non-terminating* parser.

Examples ctd.

Example (Reserved words)

A PEG grammar for recognizing identifiers that are not reserved words:

$$\begin{aligned} \texttt{identifier} &:= \texttt{!reserved letter+ ws} \\ \texttt{reserved} &:= \texttt{IF} \ / \ \ldots \\ \texttt{IF} &:= [\text{“}\textit{if}\text{”}]\ \texttt{!letter ws} \end{aligned}$$

Example (“Dangling” else)

Consider the following part of a CFG for the C language:

$$\begin{aligned} \texttt{stmt} := &\ \texttt{IF (expr) stmt} \\ |&\ \texttt{IF (expr) stmt ELSE stmt} \\ |&\ \ldots \end{aligned}$$

According to this grammar there are two possible readings of a statement

$$\text{if } (e_1) \text{ if } (e_2)\ s_1 \text{ else } s_2$$

This ambiguity is easy to resolve using PEGs.

Examples ctd.

Example (Reserved words)

A PEG grammar for recognizing identifiers that are not reserved words:

$$
\begin{array}{rcl}
\texttt{identifier} & := & \texttt{!reserved letter+ ws} \\
\texttt{reserved} & := & \texttt{IF} \;\; / \;\; \ldots \\
\texttt{IF} & := & [\text{“}\textit{if}\text{”}] \; \texttt{!letter ws}
\end{array}
$$

Example ("Dangling" else)

Consider the following part of a CFG for the C language:

$$
\begin{array}{rcl}
\texttt{stmt} & := & \texttt{IF (expr) stmt} \\
& | & \texttt{IF (expr) stmt ELSE stmt} \\
& | & \ldots
\end{array}
$$

According to this grammar there are two possible readings of a statement

$$\text{if } (e_1) \text{ if } (e_2) \; s_1 \text{ else } s_2$$

This ambiguity is easy to resolve using PEGs.

Formal definition of PEGs

Definition (PEG)

A parsing expressions grammar (PEG), $\mathcal{G}$, is a quadruple $(\mathcal{V}_N, \mathcal{V}_T, \mathrm{P}_{\mathrm{exp}}, e_s)$, where:

- $\mathcal{V}_T$ is a finite set of terminals,
- $\mathcal{V}_N$ is a finite set of non-terminals of the grammar,
- $\mathrm{P}_{\mathrm{exp}}$ is the interpretation of the productions of the grammar, *i.e.*, $\mathrm{P}_{\mathrm{exp}} : \mathcal{V}_N \to \Delta$ and
- e_s is the start production of the grammar, $e_s \in \mathcal{V}_N$.

Definition

Semantics of PEGs:

- $(e, s) \overset{m}{\rightsquigarrow} \perp$: expression e fails on input string s.
- $(e, s) \overset{m}{\rightsquigarrow} \surd_{s'}$: expression e succeeds on s, leaving s' to be parsed.

Formal definition of PEGs

Definition (PEG)

A parsing expressions grammar (PEG), $\mathcal{G}$, is a quadruple $(\mathcal{V}_N, \mathcal{V}_T, \mathrm{P}_{\mathrm{exp}}, e_s)$, where:

- $\mathcal{V}_T$ is a finite set of terminals,
- $\mathcal{V}_N$ is a finite set of non-terminals of the grammar,
- $\mathrm{P}_{\mathrm{exp}}$ is the interpretation of the productions of the grammar, *i.e.*, $\mathrm{P}_{\mathrm{exp}} : \mathcal{V}_N \to \Delta$ and
- e_s is the start production of the grammar, $e_s \in \mathcal{V}_N$.

Definition

Semantics of PEGs:

- $(e, s) \overset{m}{\rightsquigarrow} \perp$: expression e fails on input string s.
- $(e, s) \overset{m}{\rightsquigarrow} \surd_{s'}$: expression e succeeds on s, leaving s' to be parsed.

PEGs semantics

Definition (PEGs semantics, (1/2))

$$\frac{}{(\epsilon, s) \overset{1}{\rightsquigarrow} \surd_s} \qquad \frac{(\mathrm{P}_{\mathrm{exp}}(p), s) \overset{m}{\rightsquigarrow} r}{(p, s) \overset{m+1}{\rightsquigarrow} r}$$

$$\frac{}{([\cdot], x :: xs) \overset{1}{\rightsquigarrow} \surd_{xs}} \qquad \frac{}{([\cdot], []) \overset{1}{\rightsquigarrow} \perp}$$

$$\frac{}{(x, x :: xs) \overset{1}{\rightsquigarrow} \surd_{xs}} \qquad \frac{}{(x, []) \overset{1}{\rightsquigarrow} \perp}$$

$$\frac{x \neq y}{(y, x :: xs) \overset{1}{\rightsquigarrow} \perp}$$

MLstate

PEGs semantics

Definition (PEGs semantics, (1/2))

$$\frac{}{(\epsilon, s) \overset{1}{\rightsquigarrow} \surd_s} \qquad \frac{(\mathrm{P}_{\mathrm{exp}}(p), s) \overset{m}{\rightsquigarrow} r}{(p, s) \overset{m+1}{\rightsquigarrow} r}$$

$$\frac{}{([\cdot], x :: xs) \overset{1}{\rightsquigarrow} \surd_{xs}} \qquad \frac{}{([\cdot], []) \overset{1}{\rightsquigarrow} \bot}$$

$$\frac{}{(x, x :: xs) \overset{1}{\rightsquigarrow} \surd_{xs}} \qquad \frac{}{(x, []) \overset{1}{\rightsquigarrow} \bot}$$

$$\frac{x \neq y}{(y, x :: xs) \overset{1}{\rightsquigarrow} \bot}$$

PEGs semantics

Definition (PEGs semantics, (1/2))

$$\frac{}{(\epsilon, s) \overset{1}{\rightsquigarrow} \surd_s} \qquad \frac{(\mathrm{P_{exp}}(p), s) \overset{m}{\rightsquigarrow} r}{(p, s) \overset{m+1}{\rightsquigarrow} r}$$

$$\frac{}{([\cdot], x :: xs) \overset{1}{\rightsquigarrow} \surd_{xs}} \qquad \frac{}{([\cdot], []) \overset{1}{\rightsquigarrow} \perp}$$

$$\frac{}{(x, x :: xs) \overset{1}{\rightsquigarrow} \surd_{xs}} \qquad \frac{}{(x, []) \overset{1}{\rightsquigarrow} \perp}$$

$$\frac{x \neq y}{(y, x :: xs) \overset{1}{\rightsquigarrow} \perp}$$

PEGs semantics

Definition (PEGs semantics (2/2))

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \perp}{(e_1; e_2, s) \overset{m+1}{\rightsquigarrow} \perp} \qquad \frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd_{s'} \qquad (e_2, s') \overset{n}{\rightsquigarrow} r}{(e_1; e_2, s) \overset{m+n+1}{\rightsquigarrow} r}$$

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd_{s'}}{(e_1 / e_2, s) \overset{m+1}{\rightsquigarrow} \surd_{s'}} \qquad \frac{(e_1, s) \overset{m}{\rightsquigarrow} \perp \qquad (e_2, s) \overset{n}{\rightsquigarrow} r}{(e_1 / e_2, s) \overset{m+n+1}{\rightsquigarrow} r}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \surd_{s'} \qquad (e*, s') \overset{n}{\rightsquigarrow} \surd_{s''}}{(e*, s) \overset{m+n+1}{\rightsquigarrow} \surd_{s''}} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \perp}{(e*, s) \overset{m+1}{\rightsquigarrow} \surd_{s}}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \perp}{(!e, s) \overset{m+1}{\rightsquigarrow} \surd_{s}} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \surd_{s'}}{(!e, s) \overset{m+1}{\rightsquigarrow} \perp}$$

PEGs semantics

Definition (PEGs semantics (2/2))

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \bot}{(e_1; e_2, s) \overset{m+1}{\rightsquigarrow} \bot} \qquad \frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd_{s'} \qquad (e_2, s') \overset{n}{\rightsquigarrow} r}{(e_1; e_2, s) \overset{m+n+1}{\rightsquigarrow} r}$$

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd_{s'}}{(e_1 / e_2, s) \overset{m+1}{\rightsquigarrow} \surd_{s'}} \qquad \frac{(e_1, s) \overset{m}{\rightsquigarrow} \bot \qquad (e_2, s) \overset{n}{\rightsquigarrow} r}{(e_1 / e_2, s) \overset{m+n+1}{\rightsquigarrow} r}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \surd_{s'} \qquad (e*, s') \overset{n}{\rightsquigarrow} \surd_{s''}}{(e*, s) \overset{m+n+1}{\rightsquigarrow} \surd_{s''}} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \bot}{(e*, s) \overset{m+1}{\rightsquigarrow} \surd_{s}}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \bot}{(!e, s) \overset{m+1}{\rightsquigarrow} \surd_{s}} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \surd_{s'}}{(!e, s) \overset{m+1}{\rightsquigarrow} \bot}$$

PEGs semantics

Definition (PEGs semantics (2/2))

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \bot}{(e_1; e_2, s) \overset{m+1}{\rightsquigarrow} \bot} \qquad \frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd_{s'} \qquad (e_2, s') \overset{n}{\rightsquigarrow} r}{(e_1; e_2, s) \overset{m+n+1}{\rightsquigarrow} r}$$

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd_{s'}}{(e_1 / e_2, s) \overset{m+1}{\rightsquigarrow} \surd_{s'}} \qquad \frac{(e_1, s) \overset{m}{\rightsquigarrow} \bot \qquad (e_2, s) \overset{n}{\rightsquigarrow} r}{(e_1 / e_2, s) \overset{m+n+1}{\rightsquigarrow} r}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \surd_{s'} \qquad (e*, s') \overset{n}{\rightsquigarrow} \surd_{s''}}{(e*, s) \overset{m+n+1}{\rightsquigarrow} \surd_{s''}} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \bot}{(e*, s) \overset{m+1}{\rightsquigarrow} \surd_{s}}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \bot}{(!e, s) \overset{m+1}{\rightsquigarrow} \surd_{s}} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \surd_{s'}}{(!e, s) \overset{m+1}{\rightsquigarrow} \bot}$$

MLstate

PEGs in a nutshell

- Expressiveness:
  - ✓ PEGs are *unambiguous*,
  - ✓ PEGs can handle *lexical analysis*.
  - ✓ PEGs support *backtracking* and *unlimited lookahead*
  - ✗ ... but cannot easily handle *left-recursion*.
  - ✓ PEGs can parse all *LL(k) and LR(k)* languages... and more (including non-context-free grammars).
- Implementation:
  - ✓ Allow very easy implementation by means of a recursive descent parser.
  - ✗ Simple implementation can result in *exponential time* for parsing.
  - ✓ Using memoization ensures *linear time* complexity (packrat parsing).
  - ✗ ... but is expensive in terms of *memory consumption*.

PEGs in a nutshell

- Expressiveness:
  - ✓ PEGs are *unambiguous*,
  - ✓ PEGs can handle *lexical analysis*.
  - ✓ PEGs support *backtracking* and *unlimited lookahead*
  - ✗ ... but cannot easily handle *left-recursion*.
  - ✓ PEGs can parse all *LL(k) and LR(k)* languages... and more (including non-context-free grammars).
- Implementation:
  - ✓ Allow very easy implementation by means of a recursive descent parser.
  - ✗ Simple implementation can result in *exponential time* for parsing.
  - ✓ Using memoization ensures *linear time* complexity (packrat parsing),
  - ✗ ... but is expensive in terms of *memory consumption*.

MLstate

Semantic Actions

So far we can recognize whether a string belongs to the language defined by $\mathcal{G}$.

We need to be able to get a parse trace (AST).

For that we extend the type of parsing expressions Δ to a family of types Δ_α, where the index α is a type of the semantic value associated with the expression.

We also compositionally define default semantic values for all types of expressions.

And we introduce a new construct: coercion, $e[\mapsto]f$, which converts a semantic value v associated with e to $f(v)$.

Semantic Actions

So far we can recognize whether a string belongs to the language defined by $\mathcal{G}$.

We need to be able to get a parse trace (AST).

For that we extend the type of parsing expressions Δ to a family of types Δ_α, where the index α is a type of the semantic value associated with the expression.

We also compositionally define default semantic values for all types of expressions.

And we introduce a new construct: coercion, $e[\mapsto]f$, which converts a semantic value v associated with e to $f(v)$.

Semantic Actions

So far we can recognize whether a string belongs to the language defined by $\mathcal{G}$.

We need to be able to get a parse trace (AST).

For that we extend the type of parsing expressions Δ to a family of types Δ_α, where the index α is a type of the semantic value associated with the expression.

We also compositionally define default semantic values for all types of expressions.

And we introduce a new construct: coercion, $e[\mapsto]f$, which converts a semantic value v associated with e to $f(v)$.

Semantic Actions

So far we can recognize whether a string belongs to the language defined by $\mathcal{G}$.

We need to be able to get a parse trace (AST).

For that we extend the type of parsing expressions Δ to a family of types Δ_α, where the index α is a type of the semantic value associated with the expression.

We also compositionally define default semantic values for all types of expressions.

And we introduce a new construct: coercion, $e[\mapsto]f$, which converts a semantic value v associated with e to $f(v)$.

Semantic Actions

So far we can recognize whether a string belongs to the language defined by $\mathcal{G}$.

We need to be able to get a parse trace (AST).

For that we extend the type of parsing expressions Δ to a family of types Δ_α, where the index α is a type of the semantic value associated with the expression.

We also compositionally define default semantic values for all types of expressions.

And we introduce a new construct: coercion, $e[\mapsto]f$, which converts a semantic value v associated with e to $f(v)$.

Typing discipline

Definition (XPEGs typing)

$$\frac{}{\epsilon \in \Delta_{\mathrm{True}}} \qquad \frac{}{[\cdot] \in \Delta_{\mathrm{char}}}$$

$$\frac{a \in \mathcal{V}_T}{a \in \Delta_{\mathrm{char}}} \qquad \frac{A \in \mathcal{V}_N}{A \in \Delta_{\mathsf{P}_{\mathsf{type}}(A)}}$$

$$\frac{e_1 \in \Delta_\alpha \qquad e_2 \in \Delta_\beta}{e_1 ; e_2 \in \Delta_{\alpha * \beta}} \qquad \frac{e_1 \in \Delta_\alpha \qquad e_2 \in \Delta_\alpha}{e_1 / e_2 \in \Delta_\alpha}$$

$$\frac{e \in \Delta_\alpha}{e* \in \Delta_{\mathrm{list}\ \alpha}} \qquad \frac{e \in \Delta_\alpha}{!e \in \Delta_{\mathrm{True}}}$$

$$\frac{e \in \Delta_\alpha \qquad f : \alpha \to \beta}{e[\mapsto]f \in \Delta_\beta}$$

Typing discipline

Definition (XPEGs typing)

$$\frac{}{\epsilon \in \Delta_{\mathrm{True}}} \qquad \frac{}{[\cdot] \in \Delta_{\mathrm{char}}}$$

$$\frac{a \in \mathcal{V}_T}{a \in \Delta_{\mathrm{char}}} \qquad \frac{A \in \mathcal{V}_N}{A \in \Delta_{\mathsf{P}_{\mathsf{type}}(A)}}$$

$$\frac{e_1 \in \Delta_\alpha \qquad e_2 \in \Delta_\beta}{e_1; e_2 \in \Delta_{\alpha * \beta}} \qquad \frac{e_1 \in \Delta_\alpha \qquad e_2 \in \Delta_\alpha}{e_1 / e_2 \in \Delta_\alpha}$$

$$\frac{e \in \Delta_\alpha}{e* \in \Delta_{\mathrm{list}\ \alpha}} \qquad \frac{e \in \Delta_\alpha}{!e \in \Delta_{\mathrm{True}}}$$

$$\frac{e \in \Delta_\alpha \qquad f : \alpha \rightarrow \beta}{e[\mapsto]f \in \Delta_\beta}$$

Typing discipline

Definition (XPEGs typing)

$$\frac{}{\epsilon \in \Delta_{\mathrm{True}}} \qquad \frac{}{[\cdot] \in \Delta_{\mathrm{char}}}$$

$$\frac{a \in \mathcal{V}_T}{a \in \Delta_{\mathrm{char}}} \qquad \frac{A \in \mathcal{V}_N}{A \in \Delta_{\mathsf{P}_{\mathsf{type}}(A)}}$$

$$\frac{e_1 \in \Delta_\alpha \qquad e_2 \in \Delta_\beta}{e_1; e_2 \in \Delta_{\alpha * \beta}} \qquad \frac{e_1 \in \Delta_\alpha \qquad e_2 \in \Delta_\alpha}{e_1 / e_2 \in \Delta_\alpha}$$

$$\frac{e \in \Delta_\alpha}{e* \in \Delta_{\mathrm{list}\ \alpha}} \qquad \frac{e \in \Delta_\alpha}{!e \in \Delta_{\mathrm{True}}}$$

$$\frac{e \in \Delta_\alpha \qquad f : \alpha \to \beta}{e[\mapsto]f \in \Delta_\beta}$$

Typing discipline

Definition (XPEGs typing)

$$\frac{}{\epsilon \in \Delta_{\mathrm{True}}} \qquad \frac{}{[\cdot] \in \Delta_{\mathrm{char}}}$$

$$\frac{a \in \mathcal{V}_T}{a \in \Delta_{\mathrm{char}}} \qquad \frac{A \in \mathcal{V}_N}{A \in \Delta_{\mathsf{P}_{\mathsf{type}}(A)}}$$

$$\frac{e_1 \in \Delta_\alpha \qquad e_2 \in \Delta_\beta}{e_1 ; e_2 \in \Delta_{\alpha * \beta}} \qquad \frac{e_1 \in \Delta_\alpha \qquad e_2 \in \Delta_\alpha}{e_1 / e_2 \in \Delta_\alpha}$$

$$\frac{e \in \Delta_\alpha}{e* \in \Delta_{\mathrm{list}\ \alpha}} \qquad \frac{e \in \Delta_\alpha}{!e \in \Delta_{\mathrm{True}}}$$

$$\frac{e \in \Delta_\alpha \qquad f : \alpha \to \beta}{e[\mapsto]f \in \Delta_\beta}$$

Typing discipline

Definition (XPEGs typing)

$$\frac{}{\epsilon \in \Delta_{\mathrm{True}}} \qquad \frac{}{[\cdot] \in \Delta_{\mathrm{char}}}$$

$$\frac{a \in \mathcal{V}_T}{a \in \Delta_{\mathrm{char}}} \qquad \frac{A \in \mathcal{V}_N}{A \in \Delta_{\mathsf{P}_{\mathsf{type}}(A)}}$$

$$\frac{e_1 \in \Delta_\alpha \qquad e_2 \in \Delta_\beta}{e_1; e_2 \in \Delta_{\alpha * \beta}} \qquad \frac{e_1 \in \Delta_\alpha \qquad e_2 \in \Delta_\alpha}{e_1 / e_2 \in \Delta_\alpha}$$

$$\frac{e \in \Delta_\alpha}{e* \in \Delta_{\mathrm{list}\ \alpha}} \qquad \frac{e \in \Delta_\alpha}{!e \in \Delta_{\mathrm{True}}}$$

$$\frac{e \in \Delta_\alpha \qquad f : \alpha \rightarrow \beta}{e[\mapsto]f \in \Delta_\beta}$$

Derived operators

Example

The typed versions of our derived operators become:

$$
\begin{array}{rcll}
[a\text{–}z] : \Delta_{\mathrm{char}} & := & [a] \;\; / \;\; \ldots \;\; / \;\; [z] \\
[\text{“}s\text{”}] : \Delta_{\mathrm{string}} & := & [s_0]\,;\ldots;\,[s_n] \;\; [\mapsto] \;\; \text{tuple2str} \\
e+ : \Delta_{\mathrm{list}\,\alpha} & := & e; e* \;\; [\mapsto] \;\; \lambda x \,.\, x_1 :: x_2 \\
e? : \Delta_{\mathrm{option}\,\alpha} & := & e \;\; [\mapsto] \;\; \lambda x \,.\, \mathrm{Some}\; x \\
& / & \epsilon \;\; [\mapsto] \;\; \lambda x \,.\, \mathrm{None} \\
\&e : \Delta_{\mathrm{True}} & := & !!e
\end{array}
$$

where tuple2str$(c_1, \ldots, c_n) = [c_1; \ldots; c_n]$.

Example ctd.

Example (Simple mathematical expressions ctd.)

We often want to ignore the semantical values attached to an expression: $e[\sharp] := e \ [\mapsto] \ \lambda x \,.\, I$.

$$
\begin{array}{rcll}
\texttt{ws} & := & ([\textvisiblespace] \ / \ [\backslash t])* & [\sharp] \\
\texttt{number} & := & [0\text{–}9]+ & [\mapsto] \ \ \text{digListToNat} \\
\texttt{term} & := & \texttt{ws number ws} & [\mapsto] \ \ \lambda x \,.\, x_2 \\
 & / & \texttt{ws} \ [(] \ \texttt{expr} \ [)] \ \texttt{ws} & [\mapsto] \ \ \lambda x \,.\, x_3 \\
\texttt{factor} & := & \texttt{term} \ [*] \ \texttt{factor} & [\mapsto] \ \ \lambda x \,.\, x_1 * x_3 \\
 & / & \texttt{term} & \\
\texttt{expr} & := & \texttt{factor} \ [+] \ \texttt{expr} & [\mapsto] \ \ \lambda x \,.\, x_1 + x_3 \\
 & / & \texttt{factor} &
\end{array}
$$

$\Rightarrow$ [“(1+2) * (3 * 4)”] = 36

Example ctd.

Example (Simple mathematical expressions ctd.)

We often want to ignore the semantical values attached to an expression: $e[\sharp] := e \;\; [\mapsto] \;\; \lambda x \,.\, I$.

$$
\begin{array}{rcll}
\texttt{ws} & := & ([\text{␣}] \;/\; [\backslash t])* & [\sharp] \\
\texttt{number} & := & [0\text{–}9]+ & [\mapsto] \;\; \text{digListToNat} \\
\texttt{term} & := & \texttt{ws} \;\; \texttt{number} \;\; \texttt{ws} & [\mapsto] \;\; \lambda x \,.\, x_2 \\
 & / & \texttt{ws} \;\; [(] \;\; \texttt{expr} \;\; [)] \;\; \texttt{ws} & [\mapsto] \;\; \lambda x \,.\, x_3 \\
\texttt{factor} & := & \texttt{term} \;\; [*] \;\; \texttt{factor} & [\mapsto] \;\; \lambda x \,.\, x_1 * x_3 \\
 & / & \texttt{term} & \\
\texttt{expr} & := & \texttt{factor} \;\; [+] \;\; \texttt{expr} & [\mapsto] \;\; \lambda x \,.\, x_1 + x_3 \\
 & / & \texttt{factor} &
\end{array}
$$

$\Rightarrow$ [“`(1+2) * (3 * 4)`”] $= 36$

XPEGs

Definition (Extended Parsing Expressions Grammar (XPEG))

An extended parsing expressions grammar (XPEG), $\mathcal{G}$, is a tuple $(\mathcal{V}_N, \mathcal{V}_T, \mathrm{P}_{\mathsf{type}}, \mathrm{P}_{\mathsf{exp}}, v_{\mathsf{start}})$, where:

- $\mathcal{V}_T$ is a finite set of terminals,
- $\mathcal{V}_N$ is a finite set of non-terminals of the grammar,
- $\mathrm{P}_{\mathsf{type}} : \mathcal{V}_N \to \mathrm{Type}$ is a function that gives types of semantic values of all productions.
- $\mathrm{P}_{\mathsf{exp}}$ is the interpretation of the productions of the grammar, *i.e.*, $\mathrm{P}_{\mathsf{exp}} : \forall_{p:\mathcal{V}_N} \Delta_{\mathrm{P}_{\mathsf{type}}(p)}$ and
- v_{start} is the start production of the grammar, $v_{\mathsf{start}} \in \mathcal{V}_N$.

Definition (Semantics of XPEGs (1/2))

$$\frac{}{(\epsilon, s) \overset{1}{\rightsquigarrow} \surd^{I}_{s}} \qquad \frac{(\mathsf{P}_{\mathrm{exp}}(p), s) \overset{m}{\rightsquigarrow} r}{(p, s) \overset{m+1}{\rightsquigarrow} r}$$

$$\frac{}{([\cdot], x :: xs) \overset{1}{\rightsquigarrow} \surd^{x}_{xs}} \qquad \frac{}{([\cdot], []) \overset{1}{\rightsquigarrow} \bot}$$

$$\frac{}{(x, x :: xs) \overset{1}{\rightsquigarrow} \surd^{x}_{xs}} \qquad \frac{}{(x, []) \overset{1}{\rightsquigarrow} \bot}$$

$$\frac{x \neq y}{(y, x :: xs) \overset{1}{\rightsquigarrow} \bot}$$

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \bot}{(e_1; e_2, s) \overset{m+1}{\rightsquigarrow} \bot} \qquad \frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd^{v_1}_{s'} \qquad (e_2, s') \overset{n}{\rightsquigarrow} \bot}{(e_1; e_2, s) \overset{m+n+1}{\rightsquigarrow} \bot}$$

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd^{v_1}_{s'} \qquad (e_2, s') \overset{n}{\rightsquigarrow} \surd^{v_2}_{s''}}{(e_1; e_2, s) \overset{m+n+1}{\rightsquigarrow} \surd^{(v_1, v_2)}_{s''}}$$

Definition (Semantics of XPEGs (1/2))

$$\frac{}{(\epsilon, s) \overset{1}{\rightsquigarrow} \surd_s^{I}}$$

$$\frac{(\mathrm{P_{exp}}(p), s) \overset{m}{\rightsquigarrow} r}{(p, s) \overset{m+1}{\rightsquigarrow} r}$$

$$\frac{}{([\cdot], x :: xs) \overset{1}{\rightsquigarrow} \surd_{xs}^{x}}$$

$$\frac{}{([\cdot], []) \overset{1}{\rightsquigarrow} \bot}$$

$$\frac{}{(x, x :: xs) \overset{1}{\rightsquigarrow} \surd_{xs}^{x}}$$

$$\frac{}{(x, []) \overset{1}{\rightsquigarrow} \bot}$$

$$\frac{x \neq y}{(y, x :: xs) \overset{1}{\rightsquigarrow} \bot}$$

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \bot}{(e_1; e_2, s) \overset{m+1}{\rightsquigarrow} \bot}$$

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd_{s'}^{v_1} \qquad (e_2, s') \overset{n}{\rightsquigarrow} \bot}{(e_1; e_2, s) \overset{m+n+1}{\rightsquigarrow} \bot}$$

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd_{s'}^{v_1} \qquad (e_2, s') \overset{n}{\rightsquigarrow} \surd_{s''}^{v_2}}{(e_1; e_2, s) \overset{m+n+1}{\rightsquigarrow} \surd_{s''}^{(v_1, v_2)}}$$

Definition (Semantics of XPEGs (1/2))

$$\frac{}{(\epsilon, s) \overset{1}{\rightsquigarrow} \surd_s^I}$$

$$\frac{(\mathsf{P}_{\exp}(p), s) \overset{m}{\rightsquigarrow} r}{(p, s) \overset{m+1}{\rightsquigarrow} r}$$

$$\frac{}{([\cdot], x :: xs) \overset{1}{\rightsquigarrow} \surd_{xs}^x}$$

$$\frac{}{([\cdot], []) \overset{1}{\rightsquigarrow} \bot}$$

$$\frac{}{(x, x :: xs) \overset{1}{\rightsquigarrow} \surd_{xs}^x}$$

$$\frac{}{(x, []) \overset{1}{\rightsquigarrow} \bot}$$

$$\frac{x \neq y}{(y, x :: xs) \overset{1}{\rightsquigarrow} \bot}$$

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \bot}{(e_1; e_2, s) \overset{m+1}{\rightsquigarrow} \bot}$$

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd_{s'}^{v_1} \qquad (e_2, s') \overset{n}{\rightsquigarrow} \bot}{(e_1; e_2, s) \overset{m+n+1}{\rightsquigarrow} \bot}$$

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd_{s'}^{v_1} \qquad (e_2, s') \overset{n}{\rightsquigarrow} \surd_{s''}^{v_2}}{(e_1; e_2, s) \overset{m+n+1}{\rightsquigarrow} \surd_{s''}^{(v_1, v_2)}}$$

Definition (Semantics of XPEGs (2/2))

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd^{v}_{s'}}{(e_1/e_2, s) \overset{m+1}{\rightsquigarrow} \surd^{v}_{s'}} \qquad \frac{(e_1, s) \overset{m}{\rightsquigarrow} \bot \qquad (e_2, s) \overset{n}{\rightsquigarrow} r}{(e_1/e_2, s) \overset{m+n+1}{\rightsquigarrow} r}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \surd^{v}_{s'} \qquad (e*, s') \overset{n}{\rightsquigarrow} \surd^{vs}_{s''}}{(e*, s) \overset{m+n+1}{\rightsquigarrow} \surd^{v::vs}_{s''}} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \bot}{(e*, s) \overset{m+1}{\rightsquigarrow} \surd^{[]}_{s}}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \bot}{(!e, s) \overset{m+1}{\rightsquigarrow} \surd^{I}_{s}} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \surd^{v}_{s'}}{(!e, s) \overset{m+1}{\rightsquigarrow} \bot}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \bot}{(e[\mapsto]f, s) \overset{m+1}{\rightsquigarrow} \bot} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \surd^{v}_{s'}}{(e[\mapsto]f, s) \overset{m+1}{\rightsquigarrow} \surd^{f(v)}_{s'}}$$

Definition (Semantics of XPEGs (2/2))

$$\frac{(e_1, s) \overset{m}{\leadsto} \surd^{v}_{s'}}{(e_1/e_2, s) \overset{m+1}{\leadsto} \surd^{v}_{s'}} \qquad \frac{(e_1, s) \overset{m}{\leadsto} \bot \qquad (e_2, s) \overset{n}{\leadsto} r}{(e_1/e_2, s) \overset{m+n+1}{\leadsto} r}$$

$$\frac{(e, s) \overset{m}{\leadsto} \surd^{v}_{s'} \qquad (e*, s') \overset{n}{\leadsto} \surd^{vs}_{s''}}{(e*, s) \overset{m+n+1}{\leadsto} \surd^{v::vs}_{s''}} \qquad \frac{(e, s) \overset{m}{\leadsto} \bot}{(e*, s) \overset{m+1}{\leadsto} \surd^{[]}_{s}}$$

$$\frac{(e, s) \overset{m}{\leadsto} \bot}{(!e, s) \overset{m+1}{\leadsto} \surd^{I}_{s}} \qquad \frac{(e, s) \overset{m}{\leadsto} \surd^{v}_{s'}}{(!e, s) \overset{m+1}{\leadsto} \bot}$$

$$\frac{(e, s) \overset{m}{\leadsto} \bot}{(e[\mapsto]f, s) \overset{m+1}{\leadsto} \bot} \qquad \frac{(e, s) \overset{m}{\leadsto} \surd^{v}_{s'}}{(e[\mapsto]f, s) \overset{m+1}{\leadsto} \surd^{f(v)}_{s'}}$$

Definition (Semantics of XPEGs (2/2))

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd^{v}_{s'}}{(e_1/e_2, s) \overset{m+1}{\rightsquigarrow} \surd^{v}_{s'}} \qquad \frac{(e_1, s) \overset{m}{\rightsquigarrow} \bot \qquad (e_2, s) \overset{n}{\rightsquigarrow} r}{(e_1/e_2, s) \overset{m+n+1}{\rightsquigarrow} r}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \surd^{v}_{s'} \qquad (e*, s') \overset{n}{\rightsquigarrow} \surd^{vs}_{s''}}{(e*, s) \overset{m+n+1}{\rightsquigarrow} \surd^{v::vs}_{s''}} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \bot}{(e*, s) \overset{m+1}{\rightsquigarrow} \surd^{[]}_{s}}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \bot}{(!e, s) \overset{m+1}{\rightsquigarrow} \surd^{I}_{s}} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \surd^{v}_{s'}}{(!e, s) \overset{m+1}{\rightsquigarrow} \bot}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \bot}{(e[\mapsto]f, s) \overset{m+1}{\rightsquigarrow} \bot} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \surd^{v}_{s'}}{(e[\mapsto]f, s) \overset{m+1}{\rightsquigarrow} \surd^{f(v)}_{s'}}$$

MLstate

Definition (Semantics of XPEGs (2/2))

$$\frac{(e_1, s) \overset{m}{\rightsquigarrow} \surd^{v}_{s'}}{(e_1/e_2, s) \overset{m+1}{\rightsquigarrow} \surd^{v}_{s'}} \qquad \frac{(e_1, s) \overset{m}{\rightsquigarrow} \bot \qquad (e_2, s) \overset{n}{\rightsquigarrow} r}{(e_1/e_2, s) \overset{m+n+1}{\rightsquigarrow} r}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \surd^{v}_{s'} \qquad (e*, s') \overset{n}{\rightsquigarrow} \surd^{vs}_{s''}}{(e*, s) \overset{m+n+1}{\rightsquigarrow} \surd^{v::vs}_{s''}} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \bot}{(e*, s) \overset{m+1}{\rightsquigarrow} \surd^{[]}_{s}}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \bot}{(!e, s) \overset{m+1}{\rightsquigarrow} \surd^{I}_{s}} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \surd^{v}_{s'}}{(!e, s) \overset{m+1}{\rightsquigarrow} \bot}$$

$$\frac{(e, s) \overset{m}{\rightsquigarrow} \bot}{(e[\mapsto]f, s) \overset{m+1}{\rightsquigarrow} \bot} \qquad \frac{(e, s) \overset{m}{\rightsquigarrow} \surd^{v}_{s'}}{(e[\mapsto]f, s) \overset{m+1}{\rightsquigarrow} \surd^{f(v)}_{s'}}$$

Some meta-properties of XPEGs

Lemma

If $(e, s) \rightsquigarrow \surd^{v}_{s}$ *then* $(e*, s) \not\rightsquigarrow r$ *for all* r.

Theorem

If $(e, s) \overset{m}{\rightsquigarrow} \surd^{s'}_{v}$ *then* s' *is a suffix of* s.

Theorem (unambiguity)

If $(e, s) \overset{m_1}{\rightsquigarrow} r_1$ *and* $(e, s) \overset{m_2}{\rightsquigarrow} r_2$ *then* $m_1 = m_2$ *and* $r_1 = r_2$.

MLstate

Some meta-properties of XPEGs

Lemma

If $(e, s) \rightsquigarrow \surd^v_s$ *then* $(e*, s) \not\rightsquigarrow r$ *for all* r.

Theorem

If $(e, s) \overset{m}{\rightsquigarrow} \surd^{s'}_v$ *then* s' *is a suffix of* s.

Theorem (unambiguity)

If $(e, s) \overset{m_1}{\rightsquigarrow} r_1$ *and* $(e, s) \overset{m_2}{\rightsquigarrow} r_2$ *then* $m_1 = m_2$ *and* $r_1 = r_2$.

Some meta-properties of XPEGs

Lemma

If $(e, s) \rightsquigarrow \surd^{v}_{s}$ *then* $(e*, s) \not\rightsquigarrow r$ *for all* r.

Theorem

If $(e, s) \overset{m}{\rightsquigarrow} \surd^{s'}_{v}$ *then* s' *is a suffix of* s.

Theorem (unambiguity)

If $(e, s) \overset{m_1}{\rightsquigarrow} r_1$ *and* $(e, s) \overset{m_2}{\rightsquigarrow} r_2$ *then* $m_1 = m_2$ *and* $r_1 = r_2$.

Termination problem for PEGs

Ensuring termination of a PEG parser essentially essentially comes down to two problems:

1. termination of all semantic actions in $\mathcal{G}$ and
2. completness of $\mathcal{G}$ with respect to PEGs semantics.

1. with Coq we get that for free.
2. in general: undecidable
   but conservative analysis is sufficient in practice (essentially: complete)

Termination problem for PEGs

Ensuring termination of a PEG parser essentially essentially comes down to two problems:

1. termination of all semantic actions in $\mathcal{G}$ and
2. completeness of $\mathcal{G}$ with respect to PEGs semantics.

1. with Coq we get that for free.
2. in general: undecidable
   but conservative analysis is sufficient in practice (essentially: complete)

Termination problem for PEGs

Ensuring termination of a PEG parser essentially essentially comes down to two problems:

1. termination of all semantic actions in $\mathcal{G}$ and
2. completness of $\mathcal{G}$ with respect to PEGs semantics.

Example

Remember:

$$\texttt{expr} \;:=\; \texttt{expr}\;\; [+]\;\; \texttt{factor} \;\;/\;\; \texttt{factor}.$$

But also:

$$\texttt{A} \;:=\; \texttt{B} \;\;/\;\; \texttt{C}\;\; \texttt{!D}\;\; \texttt{A}$$

if B may fail and C and D may succeed, the former without consuming any input ... and it can get even worse with mutual recursion.

1. with Coq we get that for free.
2. in general: undecidable
   but conservative analysis is sufficient in practice (essentially complete).

Termination problem for PEGs

Ensuring termination of a PEG parser essentially essentially comes down to two problems:

1. termination of all semantic actions in $\mathcal{G}$ and
2. completness of $\mathcal{G}$ with respect to PEGs semantics.

Example

Remember:

$$\texttt{expr} := \texttt{expr}\ [+]\ \texttt{factor}\ /\ \texttt{factor}.$$

But also:

$$\texttt{A} := \texttt{B}\ /\ \texttt{C}\ \texttt{!D}\ \texttt{A}$$

if `B` may fail and `C` and `D` may succeed, the former without consuming any input ... and it can get even worse with mutual recursion.

1. with Coq we get that for free.
2. in general: undecidable

but conservative analysis is sufficient in practice (essentially

Termination problem for PEGs

Ensuring termination of a PEG parser essentially essentially comes down to two problems:

1. termination of all semantic actions in $\mathcal{G}$ and
2. completness of $\mathcal{G}$ with respect to PEGs semantics.

Example

Remember:

$$\texttt{expr} := \texttt{expr}\ [+]\ \texttt{factor}\ /\ \texttt{factor}.$$

But also:

$$\texttt{A} := \texttt{B}\ /\ \texttt{C}\ !\texttt{D}\ \texttt{A}$$

if B may fail and C and D may succeed, the former without consuming any input ... and it can get even worse with mutual recursion.

1. with Coq we get that for free.
2. in general: undecidable

Termination problem for PEGs

Ensuring termination of a PEG parser essentially essentially comes down to two problems:

1. termination of all semantic actions in $\mathcal{G}$ and
2. completeness of $\mathcal{G}$ with respect to PEGs semantics.

1. with Coq we get that for free.
2. in general: undecidable
   but conservative analysis is sufficient in practice (essentially: complete)

Termination problem for PEGs

Ensuring termination of a PEG parser essentially essentially comes down to two problems:

1. termination of all semantic actions in $\mathcal{G}$ and
2. completness of $\mathcal{G}$ with respect to PEGs semantics.

1. with Coq we get that for free.
2. in general: undecidable
   but conservative analysis is sufficient in practice (essentially: complete)

MLstate

Termination problem for PEGs

Ensuring termination of a PEG parser essentially essentially comes down to two problems:

1. termination of all semantic actions in $\mathcal{G}$ and
2. completness of $\mathcal{G}$ with respect to PEGs semantics.

1. with Coq we get that for free.
2. in general: undecidable
   but conservative analysis is sufficient in practice (essentially: complete)

MLstate

PEG analysis

We define three groups of properties over parsing expressions:

- "0": parsing expression can succeed without consuming any input,
- ">0": parsing expression can succeed after consuming some input,
- "$\bot$": parsing expression can fail.

PEG analysis

We define three groups of properties over parsing expressions:

- "0": parsing expression can succeed without consuming any input,
- "> 0": parsing expression can succeed after consuming some input,
- "$\bot$": parsing expression can fail.

$e \in \mathbb{P}_0$ means that expression e has property "0".

$e \in \mathbb{P}_{\geq 0} \ := \ e \in \mathbb{P}_0 \vee e \in \mathbb{P}_{>0}$.

Theorem

For arbitrary $e \in \Delta$ and $s \in \mathcal{S}$:

- *if $(e, s) \rightsquigarrow \surd^{v}_{s}$ then $e \in \mathbb{P}_0$,*
- *if $(e, s) \rightsquigarrow \surd^{v}_{s'}$ and $|s'| < |s|$ then $e \in \mathbb{P}_{>0}$ and*
- *if $(e, s) \rightsquigarrow \bot$ then $e \in \mathbb{P}_{\bot}$.*

PEG analysis

We define three groups of properties over parsing expressions:

- "0": parsing expression can succeed without consuming any input,
- ">0": parsing expression can succeed after consuming some input,
- "$\bot$": parsing expression can fail.

$e \in \mathbb{P}_0$ means that expression e has property "0".

$e \in \mathbb{P}_{\geq 0} := e \in \mathbb{P}_0 \vee e \in \mathbb{P}_{>0}$.

Theorem

For arbitrary $e \in \Delta$ and $s \in \mathcal{S}$:

- *if $(e, s) \rightsquigarrow \surd_s^v$ then $e \in \mathbb{P}_0$,*
- *if $(e, s) \rightsquigarrow \surd_{s'}^v$ and $|s'| < |s|$ then $e \in \mathbb{P}_{>0}$ and*
- *if $(e, s) \rightsquigarrow \bot$ then $e \in \mathbb{P}_\bot$.*

MLstate

Definition (PEG analysis (1/2))

$$\frac{}{\epsilon \in \mathbb{P}_0} \qquad \frac{}{[\cdot] \in \mathbb{P}_{>0}} \qquad \frac{}{[\cdot] \in \mathbb{P}_\perp} \qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathbb{P}_{>0}} \qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathbb{P}_\perp}$$

$$\frac{e \in \mathbb{P}_\perp}{e* \in \mathbb{P}_0} \qquad \frac{e \in \mathbb{P}_{>0}}{e* \in \mathbb{P}_{>0}} \qquad \frac{e \in \mathbb{P}_\perp}{!e \in \mathbb{P}_0} \qquad \frac{e \in \mathbb{P}_{\geq 0}}{!e \in \mathbb{P}_\perp}$$

$$\frac{\star \in \{0, >0, \perp\} \qquad A \in \mathcal{V}_N \qquad \mathrm{P}_{\mathrm{exp}}(A) \in \mathbb{P}_\star}{A \in \mathbb{P}_\star}$$

$$\frac{e_1 \in \mathbb{P}_\perp \vee (e_1 \in \mathbb{P}_{\geq 0} \wedge e_2 \in \mathbb{P}_\perp)}{e_1; e_2 \in \mathbb{P}_\perp}$$

$$\frac{e_1 \in \mathbb{P}_0 \qquad e_2 \in \mathbb{P}_0}{e_1; e_2 \in \mathbb{P}_0}$$

Definition (PEG analysis (1/2))

$$\frac{}{\epsilon \in \mathbb{P}_0} \qquad \frac{}{[\cdot] \in \mathbb{P}_{>0}} \qquad \frac{}{[\cdot] \in \mathbb{P}_\perp} \qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathbb{P}_{>0}} \qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathbb{P}_\perp}$$

$$\frac{e \in \mathbb{P}_\perp}{e* \in \mathbb{P}_0} \qquad \frac{e \in \mathbb{P}_{>0}}{e* \in \mathbb{P}_{>0}} \qquad \frac{e \in \mathbb{P}_\perp}{!e \in \mathbb{P}_0} \qquad \frac{e \in \mathbb{P}_{\geq 0}}{!e \in \mathbb{P}_\perp}$$

$$\frac{\star \in \{0, >0, \perp\} \qquad A \in \mathcal{V}_N \qquad \mathrm{P}_{\mathrm{exp}}(A) \in \mathbb{P}_\star}{A \in \mathbb{P}_\star}$$

$$\frac{e_1 \in \mathbb{P}_\perp \vee (e_1 \in \mathbb{P}_{\geq 0} \wedge e_2 \in \mathbb{P}_\perp)}{e_1; e_2 \in \mathbb{P}_\perp}$$

$$\frac{e_1 \in \mathbb{P}_0 \qquad e_2 \in \mathbb{P}_0}{e_1; e_2 \in \mathbb{P}_0}$$

Definition (PEG analysis (1/2))

$$\frac{}{\epsilon \in \mathbb{P}_0} \qquad \frac{}{[\cdot] \in \mathbb{P}_{>0}} \qquad \frac{}{[\cdot] \in \mathbb{P}_\bot} \qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathbb{P}_{>0}} \qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathbb{P}_\bot}$$

$$\frac{e \in \mathbb{P}_\bot}{e* \in \mathbb{P}_0} \qquad \frac{e \in \mathbb{P}_{>0}}{e* \in \mathbb{P}_{>0}} \qquad \frac{e \in \mathbb{P}_\bot}{!e \in \mathbb{P}_0} \qquad \frac{e \in \mathbb{P}_{\geq 0}}{!e \in \mathbb{P}_\bot}$$

$$\frac{\star \in \{0, >0, \bot\} \qquad A \in \mathcal{V}_N \qquad \mathrm{P}_{\mathrm{exp}}(A) \in \mathbb{P}_\star}{A \in \mathbb{P}_\star}$$

$$\frac{e_1 \in \mathbb{P}_\bot \vee (e_1 \in \mathbb{P}_{\geq 0} \wedge e_2 \in \mathbb{P}_\bot)}{e_1; e_2 \in \mathbb{P}_\bot}$$

$$\frac{e_1 \in \mathbb{P}_0 \qquad e_2 \in \mathbb{P}_0}{e_1; e_2 \in \mathbb{P}_0}$$

Definition (PEG analysis (1/2))

$$\frac{}{\epsilon \in \mathbb{P}_0} \qquad \frac{}{[\cdot] \in \mathbb{P}_{>0}} \qquad \frac{}{[\cdot] \in \mathbb{P}_{\bot}} \qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathbb{P}_{>0}} \qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathbb{P}_{\bot}}$$

$$\frac{e \in \mathbb{P}_{\bot}}{e* \in \mathbb{P}_0} \qquad \frac{e \in \mathbb{P}_{>0}}{e* \in \mathbb{P}_{>0}} \qquad \frac{e \in \mathbb{P}_{\bot}}{!e \in \mathbb{P}_0} \qquad \frac{e \in \mathbb{P}_{\geq 0}}{!e \in \mathbb{P}_{\bot}}$$

$$\frac{\star \in \{0, >0, \bot\} \qquad A \in \mathcal{V}_N \qquad \mathrm{P}_{\mathrm{exp}}(A) \in \mathbb{P}_\star}{A \in \mathbb{P}_\star}$$

$$\frac{e_1 \in \mathbb{P}_{\bot} \vee (e_1 \in \mathbb{P}_{\geq 0} \wedge e_2 \in \mathbb{P}_{\bot})}{e_1 ; e_2 \in \mathbb{P}_{\bot}}$$

$$\frac{e_1 \in \mathbb{P}_0 \qquad e_2 \in \mathbb{P}_0}{e_1 ; e_2 \in \mathbb{P}_0}$$

MLstate

Definition (PEG analysis (2/2))

$$\frac{(e_1 \in \mathbb{P}_{>0} \wedge e_2 \in \mathbb{P}_{\geq 0}) \vee (e_1 \in \mathbb{P}_{\geq 0} \wedge e_2 \in \mathbb{P}_{>0})}{e_1 ; e_2 \in \mathbb{P}_{>0}}$$

$$\frac{e_1 \in \mathbb{P}_{\perp} \qquad e_2 \in \mathbb{P}_{\perp}}{e_1 / e_2 \in \mathbb{P}_{\perp}}$$

$$\frac{\star \in \{0, > 0\} \qquad e_1 \in \mathbb{P}_{\star} \vee (e_1 \in \mathbb{P}_{\perp} \wedge e_2 \in \mathbb{P}_{\star})}{e_1 / e_2 \in \mathbb{P}_{\star}}$$

Definition (Expression set of $\mathcal{G}$)

$$E(\mathcal{G}) = \{e' \mid e' \sqsubseteq e, e \in P_{\exp}(A), A \in \mathcal{V}_N\}$$

$\Rightarrow$ We compute those properties by iterating over $E(\mathcal{G})$ from $\emptyset$ until reaching a fixpoint.

MLstate

Definition (PEG analysis (2/2))

$$\frac{(e_1 \in \mathbb{P}_{>0} \land e_2 \in \mathbb{P}_{\geq 0}) \lor (e_1 \in \mathbb{P}_{\geq 0} \land e_2 \in \mathbb{P}_{>0})}{e_1 ; e_2 \in \mathbb{P}_{>0}}$$

$$\frac{e_1 \in \mathbb{P}_{\perp} \qquad e_2 \in \mathbb{P}_{\perp}}{e_1 / e_2 \in \mathbb{P}_{\perp}}$$

$$\frac{\star \in \{0, > 0\} \qquad e_1 \in \mathbb{P}_{\star} \lor (e_1 \in \mathbb{P}_{\perp} \land e_2 \in \mathbb{P}_{\star})}{e_1 / e_2 \in \mathbb{P}_{\star}}$$

Definition (Expression set of $\mathcal{G}$)

$$\mathrm{E}(\mathcal{G}) = \{e' \mid e' \sqsubseteq e, e \in \mathrm{P}_{\exp}(A), A \in \mathcal{V}_N\}$$

$\Rightarrow$ We compute those properties by iterating over $\mathrm{E}(\mathcal{G})$ from $\emptyset$ until reaching a fixpoint.

MLstate

Definition (PEG analysis (2/2))

$$\frac{(e_1 \in \mathbb{P}_{>0} \wedge e_2 \in \mathbb{P}_{\geq 0}) \vee (e_1 \in \mathbb{P}_{\geq 0} \wedge e_2 \in \mathbb{P}_{>0})}{e_1 ; e_2 \in \mathbb{P}_{>0}}$$

$$\frac{e_1 \in \mathbb{P}_{\perp} \qquad e_2 \in \mathbb{P}_{\perp}}{e_1 / e_2 \in \mathbb{P}_{\perp}}$$

$$\frac{\star \in \{0, > 0\} \qquad e_1 \in \mathbb{P}_{\star} \vee (e_1 \in \mathbb{P}_{\perp} \wedge e_2 \in \mathbb{P}_{\star})}{e_1 / e_2 \in \mathbb{P}_{\star}}$$

Definition (Expression set of $\mathcal{G}$)

$$\mathrm{E}(\mathcal{G}) = \{e' \mid e' \sqsubseteq e, e \in \mathrm{P}_{\mathrm{exp}}(A), A \in \mathcal{V}_N\}$$

$\Rightarrow$ We compute those properties by iterating over $\mathrm{E}(\mathcal{G})$ from $\emptyset$ until reaching a fixpoint.

Definition (PEG analysis (2/2))

$$\frac{(e_1 \in \mathbb{P}_{>0} \wedge e_2 \in \mathbb{P}_{\geq 0}) \vee (e_1 \in \mathbb{P}_{\geq 0} \wedge e_2 \in \mathbb{P}_{>0})}{e_1 ; e_2 \in \mathbb{P}_{>0}}$$

$$\frac{e_1 \in \mathbb{P}_{\perp} \qquad e_2 \in \mathbb{P}_{\perp}}{e_1 / e_2 \in \mathbb{P}_{\perp}}$$

$$\frac{\star \in \{0, >0\} \qquad e_1 \in \mathbb{P}_{\star} \vee (e_1 \in \mathbb{P}_{\perp} \wedge e_2 \in \mathbb{P}_{\star})}{e_1 / e_2 \in \mathbb{P}_{\star}}$$

Definition (Expression set of $\mathcal{G}$)

$$\mathrm{E}(\mathcal{G}) = \{e' \mid e' \sqsubseteq e, e \in \mathrm{P}_{\mathrm{exp}}(A), A \in \mathcal{V}_N\}$$

$\Rightarrow$ We compute those properties by iterating over $\mathrm{E}(\mathcal{G})$ from $\emptyset$ until reaching a fixpoint.

Definition (Checking WF)

$$\frac{e_1 \in \mathrm{WF} \qquad e_2 \in \mathrm{WF}}{e_1 / e_2 \in \mathrm{WF}} \qquad\qquad \frac{}{\epsilon \in \mathrm{WF}}$$

$$\frac{A \in \mathcal{V}_N \qquad \mathrm{P}_{\mathrm{exp}}(A) \in \mathrm{WF}}{A \in \mathrm{WF}} \qquad\qquad \frac{}{[\cdot] \in \mathrm{WF}}$$

$$\frac{e_1 \in \mathrm{WF} \qquad e_1 \in \mathbb{P}_0 \Rightarrow e_2 \in \mathrm{WF}}{e_1 ; e_2 \in \mathrm{WF}} \qquad\qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathrm{WF}}$$

$$\frac{e \in \mathrm{WF}, \qquad e \notin \mathbb{P}_0}{e* \in \mathrm{WF}} \qquad\qquad \frac{e \in \mathrm{WF}}{!e \in \mathrm{WF}}$$

$\Rightarrow$ Again iteration from $\emptyset$ until reaching a fixpoint

Definition (Checking WF)

$$\frac{e_1 \in \mathsf{WF} \qquad e_2 \in \mathsf{WF}}{e_1 / e_2 \in \mathsf{WF}} \qquad \frac{}{\epsilon \in \mathsf{WF}}$$

$$\frac{A \in \mathcal{V}_N \qquad \mathsf{P}_{\mathrm{exp}}(A) \in \mathsf{WF}}{A \in \mathsf{WF}} \qquad \frac{}{[\cdot] \in \mathsf{WF}}$$

$$\frac{e_1 \in \mathsf{WF} \qquad e_1 \in \mathbb{P}_0 \Rightarrow e_2 \in \mathsf{WF}}{e_1 ; e_2 \in \mathsf{WF}} \qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathsf{WF}}$$

$$\frac{e \in \mathsf{WF}, \quad e \notin \mathbb{P}_0}{e* \in \mathsf{WF}} \qquad \frac{e \in \mathsf{WF}}{!e \in \mathsf{WF}}$$

$\Rightarrow$ Again iteration from $\emptyset$ until reaching a fixpoint

Definition (Checking WF)

$$\frac{e_1 \in \mathrm{WF} \qquad e_2 \in \mathrm{WF}}{e_1 / e_2 \in \mathrm{WF}} \qquad \frac{}{\epsilon \in \mathrm{WF}}$$

$$\frac{A \in \mathcal{V}_N \qquad \mathrm{P_{exp}}(A) \in \mathrm{WF}}{A \in \mathrm{WF}} \qquad \frac{}{[\cdot] \in \mathrm{WF}}$$

$$\frac{e_1 \in \mathrm{WF} \qquad e_1 \in \mathbb{P}_0 \Rightarrow e_2 \in \mathrm{WF}}{e_1 ; e_2 \in \mathrm{WF}} \qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathrm{WF}}$$

$$\frac{e \in \mathrm{WF}, \qquad e \notin \mathbb{P}_0}{e* \in \mathrm{WF}} \qquad \frac{e \in \mathrm{WF}}{!e \in \mathrm{WF}}$$

$\Rightarrow$ Again iteration from $\emptyset$ until reaching a fixpoint

Definition (Checking WF)

$$\frac{e_1 \in \mathrm{WF} \qquad e_2 \in \mathrm{WF}}{e_1 / e_2 \in \mathrm{WF}} \qquad \frac{}{\epsilon \in \mathrm{WF}}$$

$$\frac{A \in \mathcal{V}_N \qquad \mathrm{P}_{\mathrm{exp}}(A) \in \mathrm{WF}}{A \in \mathrm{WF}} \qquad \frac{}{[\cdot] \in \mathrm{WF}}$$

$$\frac{e_1 \in \mathrm{WF} \qquad e_1 \in \mathbb{P}_0 \Rightarrow e_2 \in \mathrm{WF}}{e_1 ; e_2 \in \mathrm{WF}} \qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathrm{WF}}$$

$$\frac{e \in \mathrm{WF}, \qquad e \notin \mathbb{P}_0}{e* \in \mathrm{WF}} \qquad \frac{e \in \mathrm{WF}}{!e \in \mathrm{WF}}$$

$\Rightarrow$ Again iteration from $\emptyset$ until reaching a fixpoint

Definition (Checking WF)

$$\frac{e_1 \in \mathsf{WF} \qquad e_2 \in \mathsf{WF}}{e_1 / e_2 \in \mathsf{WF}} \qquad \frac{}{\epsilon \in \mathsf{WF}}$$

$$\frac{A \in \mathcal{V}_N \qquad \mathsf{P}_{\mathsf{exp}}(A) \in \mathsf{WF}}{A \in \mathsf{WF}} \qquad \frac{}{[\cdot] \in \mathsf{WF}}$$

$$\frac{e_1 \in \mathsf{WF} \qquad e_1 \in \mathbb{P}_0 \Rightarrow e_2 \in \mathsf{WF}}{e_1 ; e_2 \in \mathsf{WF}} \qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathsf{WF}}$$

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

$\Rightarrow$ Again iteration from $\emptyset$ until reaching a fixpoint

Definition (Checking WF)

$$\frac{e_1 \in \mathrm{WF} \qquad e_2 \in \mathrm{WF}}{e_1 / e_2 \in \mathrm{WF}} \qquad \frac{}{\epsilon \in \mathrm{WF}}$$

$$\frac{A \in \mathcal{V}_N \qquad \mathrm{P}_{\mathrm{exp}}(A) \in \mathrm{WF}}{A \in \mathrm{WF}} \qquad \frac{}{[\cdot] \in \mathrm{WF}}$$

$$\frac{e_1 \in \mathrm{WF} \qquad e_1 \in \mathbb{P}_0 \Rightarrow e_2 \in \mathrm{WF}}{e_1 ; e_2 \in \mathrm{WF}} \qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathrm{WF}}$$

$$\frac{e \in \mathrm{WF}, \qquad e \notin \mathbb{P}_0}{e* \in \mathrm{WF}} \qquad \frac{e \in \mathrm{WF}}{!e \in \mathrm{WF}}$$

$\Rightarrow$ Again iteration from $\emptyset$ until reaching a fixpoint

Definition (Checking WF)

$$\frac{e_1 \in \mathsf{WF} \qquad e_2 \in \mathsf{WF}}{e_1 / e_2 \in \mathsf{WF}} \qquad \frac{}{\epsilon \in \mathsf{WF}}$$

$$\frac{A \in \mathcal{V}_N \qquad \mathsf{P}_{\mathsf{exp}}(A) \in \mathsf{WF}}{A \in \mathsf{WF}} \qquad \frac{}{[\cdot] \in \mathsf{WF}}$$

$$\frac{e_1 \in \mathsf{WF} \qquad e_1 \in \mathbb{P}_0 \Rightarrow e_2 \in \mathsf{WF}}{e_1 ; e_2 \in \mathsf{WF}} \qquad \frac{a \in \mathcal{V}_T}{[a] \in \mathsf{WF}}$$

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

$\Rightarrow$ Again iteration from $\emptyset$ until reaching a fixpoint

Well-formedness of (X)PEGs

Definition (WF)

We say that $\mathcal{G}$ is well-formed if $\mathrm{E}(\mathcal{G}) = \mathrm{WF}$.

Theorem

If $\mathcal{G}$ is well-formed then it is complete.

Well-formedness of (X)PEGs

Definition (WF)

We say that $\mathcal{G}$ is well-formed if $\mathsf{E}(\mathcal{G}) = \mathsf{WF}$.

Theorem

If $\mathcal{G}$ is well-formed then it is complete.

Expressing PEGs in Coq

Example

$$Parameter\ prods : Enumeration.$$
$$Parameter\ prods\_type : prods \to Type.$$

$$\textbf{Inductive}\ PExp : Type \to Type :=$$
$$\mid Terminal : char \to PExp\ char$$
$$\mid NonTerminal : \forall\ p, PExp\ (prods\_type\ p)$$
$$\mid Star : \forall\ A, PExp\ A \to PExp\ (list\ A)$$
$$\mid Choice : \forall\ A, PExp\ A \to PExp\ A \to PExp\ A$$
$$\mid \ldots$$

$\Rightarrow$ Every such PEG is well-defined and well-typed.

Expressing PEGs in Coq

Example

$$\textit{Parameter prods} : \textit{Enumeration}.$$
$$\textit{Parameter prods\_type} : \textit{prods} \to \textit{Type}.$$

$$
\begin{array}{l}
\textbf{Inductive}\ \textit{PExp} : \textit{Type} \to \textit{Type} := \\
\ \mid \textit{Terminal} : \textit{char} \to \textit{PExp char} \\
\ \mid \textit{NonTerminal} : \forall\ p, \textit{PExp}\ (\textit{prods\_type}\ p) \\
\ \mid \textit{Star} : \forall\ A, \textit{PExp}\ A \to \textit{PExp}\ (\textit{list}\ A) \\
\ \mid \textit{Choice} : \forall\ A, \textit{PExp}\ A \to \textit{PExp}\ A \to \textit{PExp}\ A \\
\ \mid \ldots
\end{array}
$$

$\Rightarrow$ Every such PEG is well-defined and well-typed.

PEGs in Coq

Example (Coq's grammar of our leading example)

Program Definition *production p* :=
  match *p* **return** *PExp* (*prod_type p*) **with**

$$
\begin{array}{llll}
\mid \mathit{ws} & \Rightarrow (\texttt{" "} \,/\, \texttt{"\textbackslash t"})\,[*] & & [\#] \\
\mid \mathit{number} & \Rightarrow [\texttt{"0"} -- \texttt{"9"}]\,[+] & & [\rightarrow]\ \mathit{digListToRat} \\
\mid \mathit{term} & \Rightarrow \mathit{ws}; \mathit{number}; \mathit{ws} & & [\rightarrow]\ (\lambda v \Rightarrow A2\_3\ v) \\
 & \quad / \ \mathit{ws}; \texttt{"("}; \mathit{expr}; \texttt{")"}; \mathit{ws} & & [\rightarrow]\ (\lambda v \Rightarrow A3\_5\ v) \\
\mid \mathit{factor} & \Rightarrow \mathit{term}; \texttt{"*"}; \mathit{factor} & & [\rightarrow]\ (\lambda v \Rightarrow A1\_3\ v * A3\_3\ v) \\
 & \quad / \ \mathit{term} & & \\
\mid \mathit{expr} & \Rightarrow \mathit{factor}; \texttt{"+"}; \mathit{expr} & & [\rightarrow]\ (\lambda v \Rightarrow A1\_3\ v + A3\_3\ v) \\
 & \quad / \ \mathit{factor} & &
\end{array}
$$

  end.

⇒ Thanks to notations and coercions this is not too different from what we had before...

Example (Coq's grammar of our leading example)

$$
\begin{array}{l}
\textbf{Program Definition } \textit{production } p := \\
\quad \textbf{match } p \textbf{ return } \textit{PExp } (\textit{prod\_type } p) \textbf{ with} \\
\quad \mid \textit{ws} \Rightarrow (\texttt{" "} / \texttt{"\textbackslash t"})\,[*] \qquad [\#] \\
\quad \mid \textit{number} \Rightarrow [\texttt{"0"} -- \texttt{"9"}]\,[+] \quad [\rightarrow]\ \textit{digListToRat} \\
\quad \mid \textit{term} \Rightarrow \textit{ws}; \textit{number}; \textit{ws} \qquad [\rightarrow]\ (\lambda v \Rightarrow A2\_3\ v) \\
\qquad\qquad / \ \textit{ws}; \texttt{"("}; \textit{expr}; \texttt{")"}; \textit{ws}\ [\rightarrow]\ (\lambda v \Rightarrow A3\_5\ v) \\
\quad \mid \textit{factor} \Rightarrow \textit{term}; \texttt{"*"}; \textit{factor} \qquad [\rightarrow]\ (\lambda v \Rightarrow A1\_3\ v * A3\_3\ v) \\
\qquad\qquad / \ \textit{term} \\
\quad \mid \textit{expr} \Rightarrow \textit{factor}; \texttt{"+"}; \textit{expr} \qquad [\rightarrow]\ (\lambda v \Rightarrow A1\_3\ v + A3\_3\ v) \\
\qquad\qquad / \ \textit{factor} \\
\quad \textbf{end}.
\end{array}
$$

⇒ Thanks to notations and coercions this is not too different from what we had before...

Checking PEGs well-formedness in Coq

Example

Program Fixpoint $\mathit{wf\_compute}\ (\mathit{wf} : \mathit{PES.t} \mid \mathit{wf\_prop}\ \mathit{wf})$
$\quad\{$**measure** $(\mathit{wf\_measure}\ \mathit{wf})\}$:
$\quad\{\mathit{wf} : \mathit{PES.t} \mid \mathit{wf\_prop}\ \mathit{wf}\} :=$
let $\mathit{wf}' := \mathit{wf\_derive}\ \mathit{wf}$ **in**
match $\mathit{PES.equal}\ \mathit{wf}\ \mathit{wf}'$ **with**
$\mid \mathit{true} \Rightarrow \mathit{wf}$
$\mid \mathit{false} \Rightarrow \mathit{wf\_compute}\ \mathit{wf}'$
end.

Theorem

- $\mathit{wf} \subseteq \mathrm{E}(\mathcal{G}) \implies \mathit{wf\_derive}\ \mathit{wf} \subseteq \mathrm{E}(\mathcal{G})$
- $\mathit{wf} \subseteq \mathit{wf\_derive}\ \mathit{wf}$.

Checking PEGs well-formedness in Coq

Example

$$
\begin{array}{l}
\textbf{Program Fixpoint } \mathit{wf\_compute}\ (\mathit{wf} : \mathit{PES.t} \mid \mathit{wf\_prop}\ \mathit{wf}) \\
\quad \{\textbf{measure } (\mathit{wf\_measure}\ \mathit{wf})\} : \\
\quad \{\mathit{wf} : \mathit{PES.t} \mid \mathit{wf\_prop}\ \mathit{wf}\} := \\
\quad \textbf{let } \mathit{wf}' := \mathit{wf\_derive}\ \mathit{wf} \textbf{ in} \\
\quad \textbf{match } \mathit{PES.equal}\ \mathit{wf}\ \mathit{wf}' \textbf{ with} \\
\quad \mid \mathit{true} \Rightarrow \mathit{wf} \\
\quad \mid \mathit{false} \Rightarrow \mathit{wf\_compute}\ \mathit{wf}' \\
\quad \textbf{end}.
\end{array}
$$

Theorem

- $\mathit{wf} \subseteq \mathrm{E}(\mathcal{G}) \implies \mathit{wf\_derive}\ \mathit{wf} \subseteq \mathrm{E}(\mathcal{G})$
- $\mathit{wf} \subseteq \mathit{wf\_derive}\ \mathit{wf}$.

PEG parser

Example

Program Fixpoint *parse*
$(T : Type)\ (e : PExp\ T \mid is\_grammar\_exp\ e)\ (s : string)$
$: \{r : ParsingResult\ T \mid \exists\ n, [e, s] \Rightarrow [n, r]\} :=$

match *e* **with**

| ...
end.

MLstate

PEG parser

Example

Program Fixpoint *parse*
  $(T : Type)\ (e : PExp\ T \mid is\_grammar\_exp\ e)\ (s : string)$
$: \{r : ParsingResult\ T \mid \exists\ n, [e, s] \Rightarrow [n, r]\} :=$

```
match e with
| Terminal c ⇒
  match s with
  | nil ⇒ Fail
  | x :: xs ⇒
    match CharAscii.eq_dec c x with
    | left _ ⇒ Ok xs c
    | right _ ⇒ Fail
    end
  end
| ...
end.
```

PEG parser

Example

Program Fixpoint *parse*

$(T : Type)\ (e : PExp\ T \mid is\_grammar\_exp\ e)\ (s : string)$

$: \{r : ParsingResult\ T \mid \exists\ n, [e, s] \Rightarrow [n, r]\} :=$

match e **with**

$\mid NonTerminal\ p \Rightarrow$

$parse\ (production\ p)\ s$

$\mid \ldots$

end.

MLstate

PEG parser

Example

Program Fixpoint *parse*

$(T : Type)\ (e : PExp\ T \mid is\_grammar\_exp\ e)\ (s : string)$

$: \{r : ParsingResult\ T \mid \exists\ n, [e, s] \Rightarrow [n, r]\} :=$

match e **with**

$\mid$ $Choice\ \_\ e1\ e2 \Rightarrow$

  match $parse\ e1\ s$ **with**

  $\mid$ $PR\_ok\ s'\ v \Rightarrow Ok\ s'\ v$

  $\mid$ $PR\_fail \Rightarrow parse\ e2\ s$

  end

$\mid \ldots$

end.

PEG parser

Example

Program Fixpoint *parse*
$(T : Type)\ (e : PExp\ T \mid is\_grammar\_exp\ e)\ (s : string)$
$: \{r : ParsingResult\ T \mid \exists\ n, [e, s] \Rightarrow [n, r]\} :=$

match *e* **with**
| *Star* _ *e* ⇒
  match *parse e s* **with**
  | $PR\_fail \Rightarrow Ok\ s\ []$
  | $PR\_ok\ s'\ v \Rightarrow$
    match *parse* $(e\ [*])\ s'$ **with**
    | $PR\_fail \Rightarrow Fail$
    | $PR\_ok\ s''\ v' \Rightarrow Ok\ s''\ (v :: v')$
    end
  end
| ...
end.

PEG parser

Example

Program Fixpoint *parse*
$(T : Type)\ (e : PExp\ T \mid is\_grammar\_exp\ e)\ (s : string)$
$: \{r : ParsingResult\ T \mid \exists\ n, [e, s] \Rightarrow [n, r]\} :=$
match *e* **with**
$\mid \ldots$
end.

How about termination?

PEG parser

Example

$$(e_1, s_1) \succ (e_2, s_2) \iff \exists_{n_1, r_1, n_2, r_2} (e_1, s_1) \overset{n_1}{\rightsquigarrow} r_1 \wedge (e_2, s_2) \overset{n_2}{\rightsquigarrow} r_2 \wedge n_1 > n_2$$

Program Fixpoint *parse*
  $(T : Type)\ (e : PExp\ T \mid is\_grammar\_exp\ e)\ (s : string)$
  $\{$**measure** $(e, s)\ (\succ)\}$
$: \{r : ParsingResult\ T \mid \exists\ n, [e, s] \Rightarrow [n, r]\} :=$
match e **with**
| ...
end.

How about termination?

PEG parser

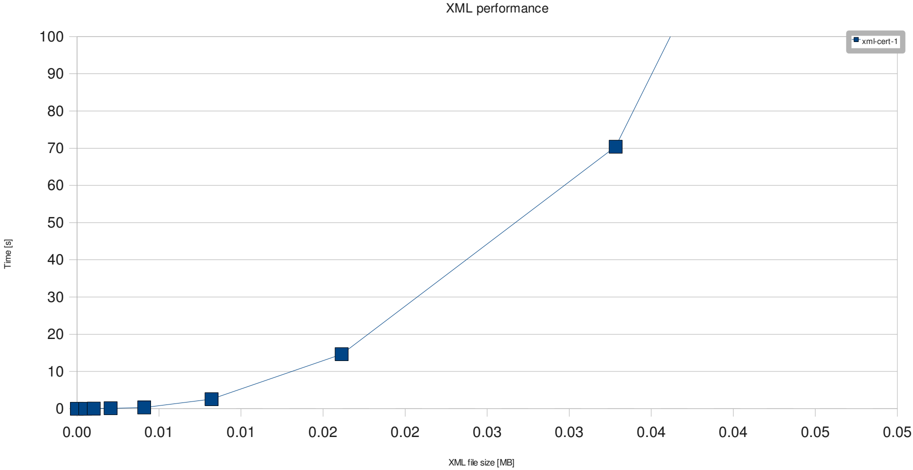
Example

Program Fixpoint *parse*
  $(T : Type)\ (e : PExp\ T \mid is\_grammar\_exp\ e)\ (s : string)$
  $\{$**measure** $(e, s)\ (\succ)\}$
$: \{r : ParsingResult\ \ T \mid \exists\ n, [e, s] \Rightarrow [n, r]\} :=$
match *e* **with**
$\mid$ ...
end.

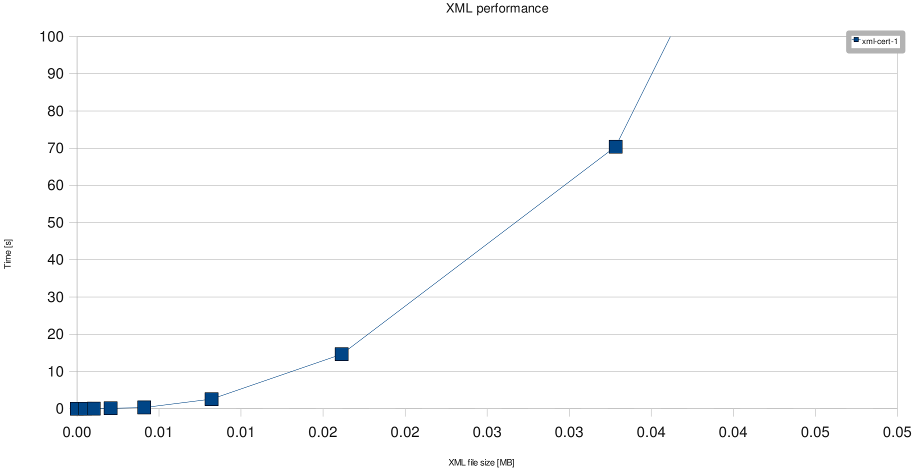
$+$ 43 TCCs.

MLstate

Evolution of TRX's performance

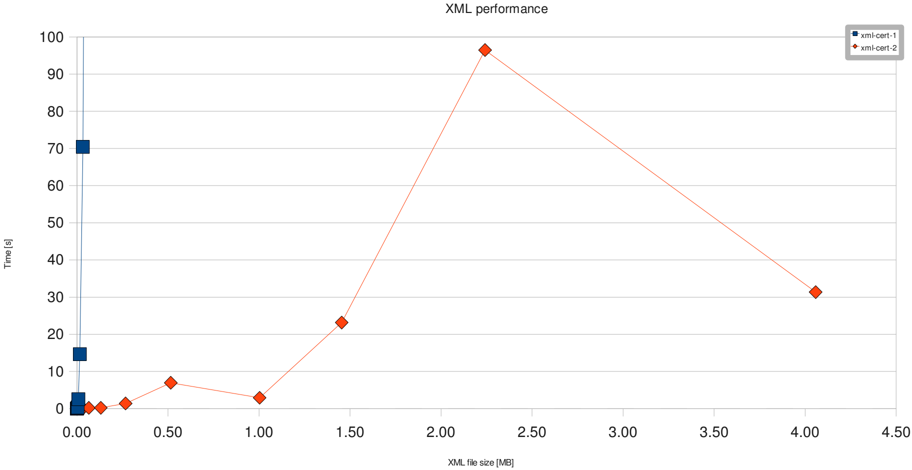


Evolution of TRX's performance

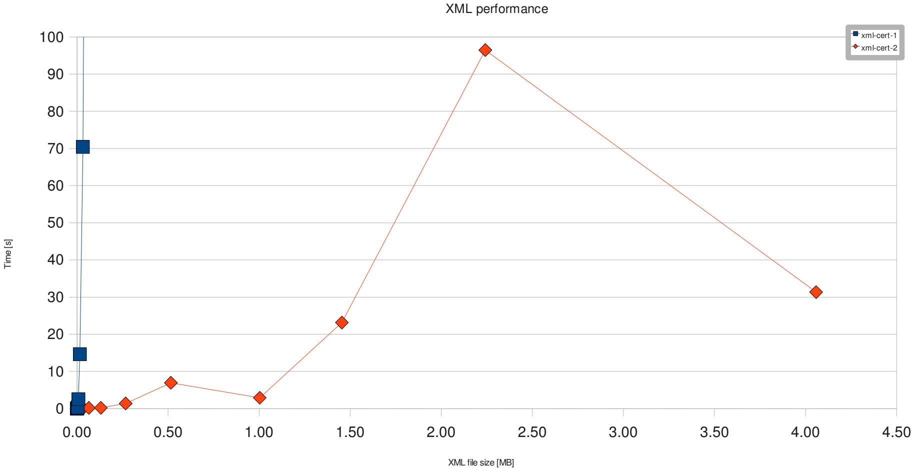


⇒ `rev` uses append at every step and hence is quadratic

Evolution of TRX's performance

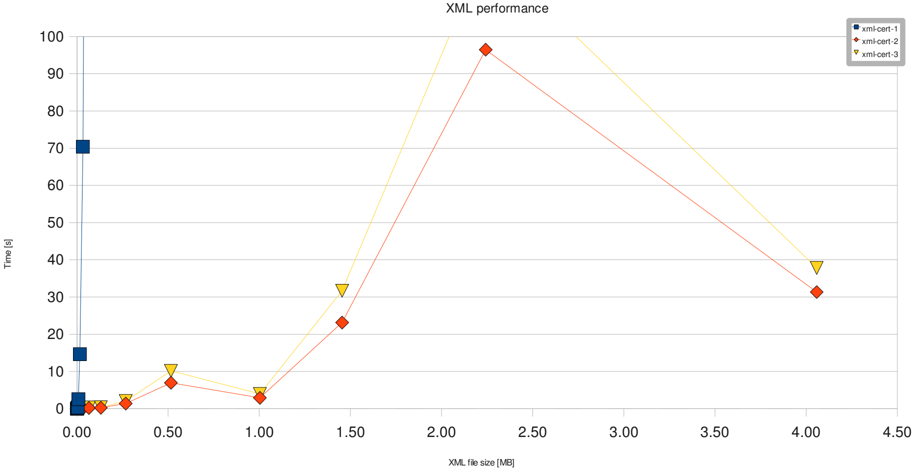


Evolution of TRX's performance



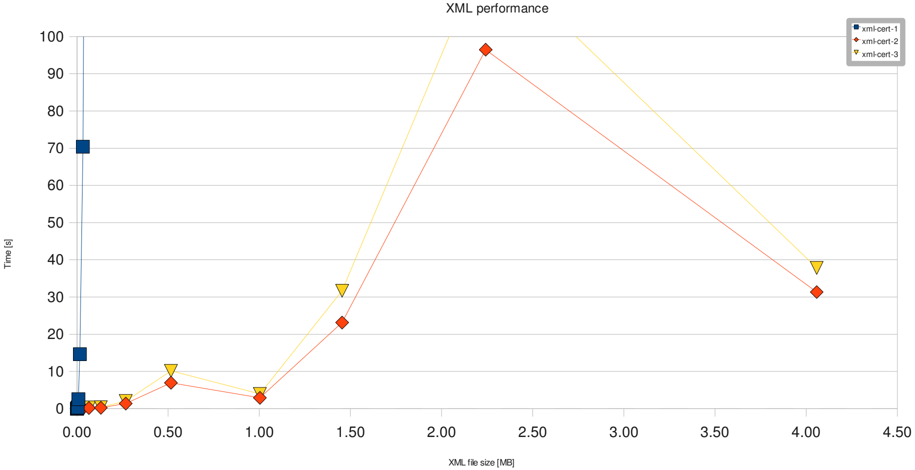
⇒ We better implement [a–z] as a primitive

Evolution of TRX's performance



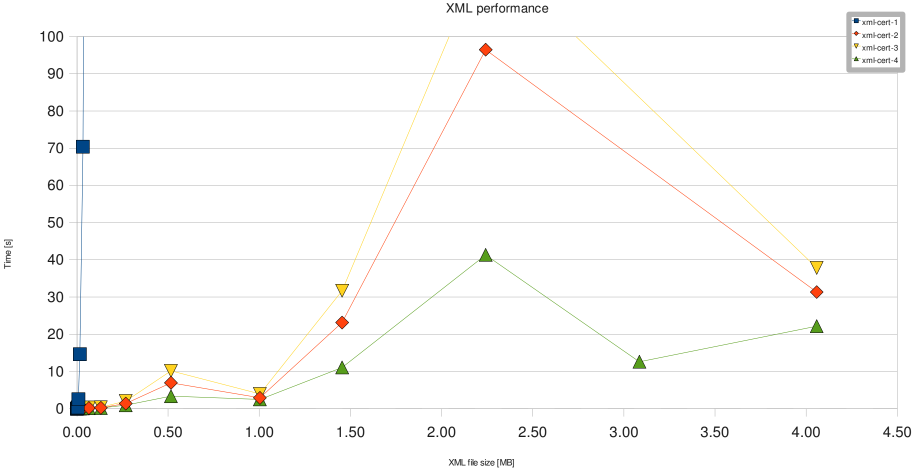
MLstate

Evolution of TRX's performance

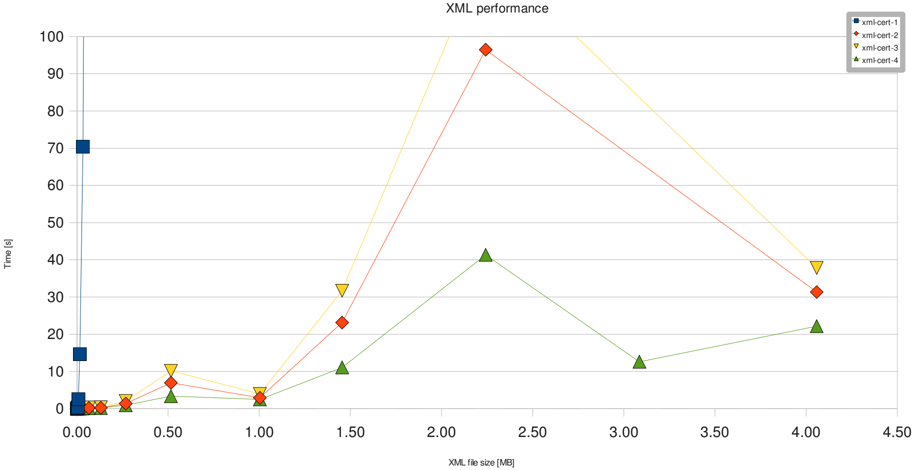


⇒ ... and we better use binary $\mathbb{N}$ to compare characters

Evolution of TRX's performance

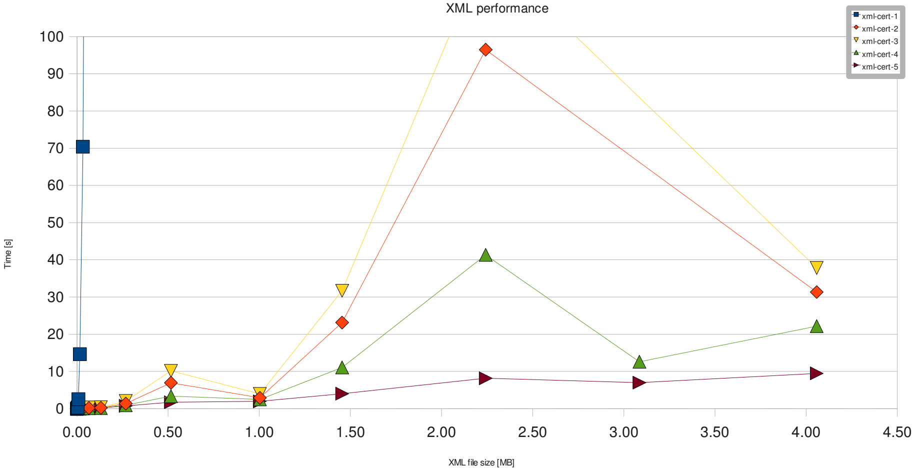


Evolution of TRX's performance

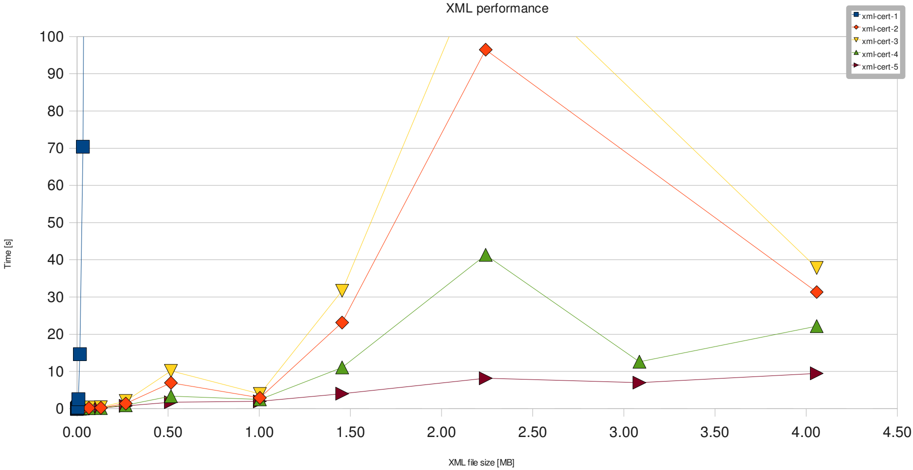


⇒ 85% of the time the parser is busy with GC! Let us tweak it

Evolution of TRX's performance

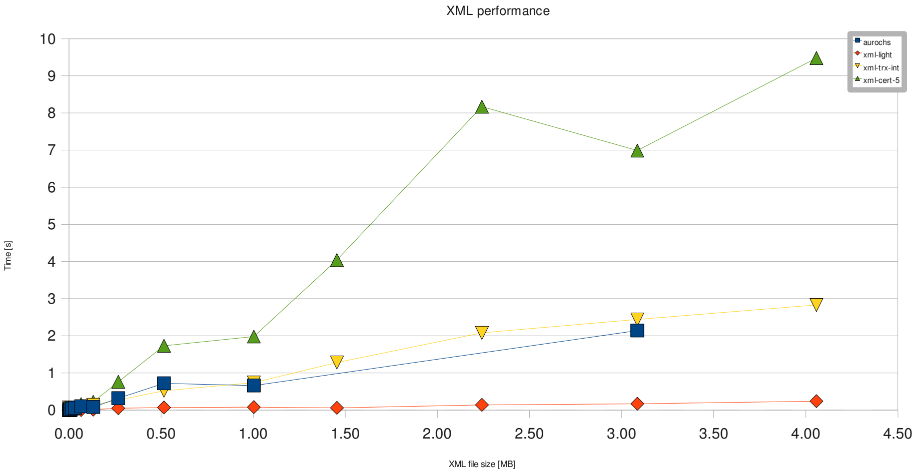


Evolution of TRX's performance



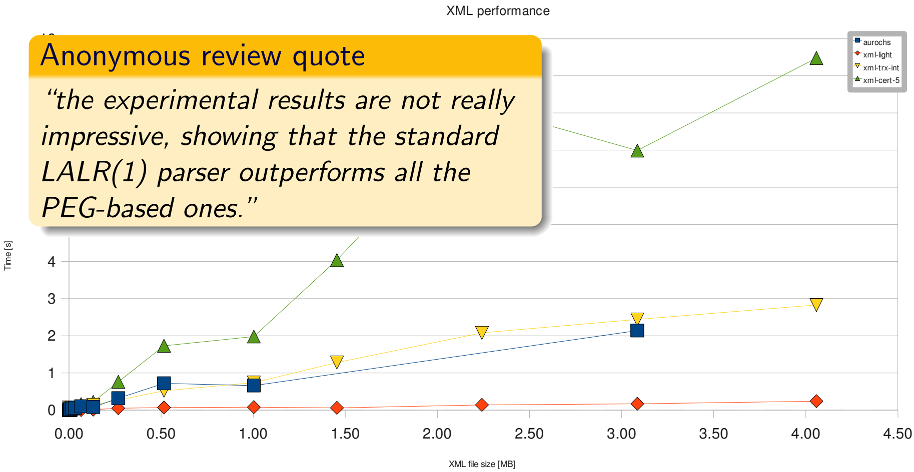
⇒ Can we do better than that?

TRX: performance comparison with other tools



- 7x slower than Aurochs (but more robust?).
- 32x slower than xml-light.

TRX: performance comparison with other tools

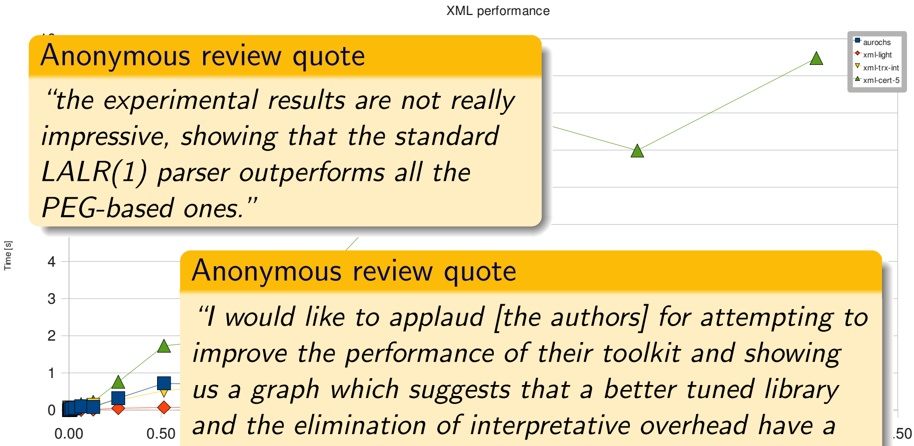


Anonymous review quote

"the experimental results are not really impressive, showing that the standard LALR(1) parser outperforms all the PEG-based ones."

- 7x slower than Aurochs (but more robust?).
- 32x slower than xml-light.

TRX: performance comparison with other tools



Anonymous review quote

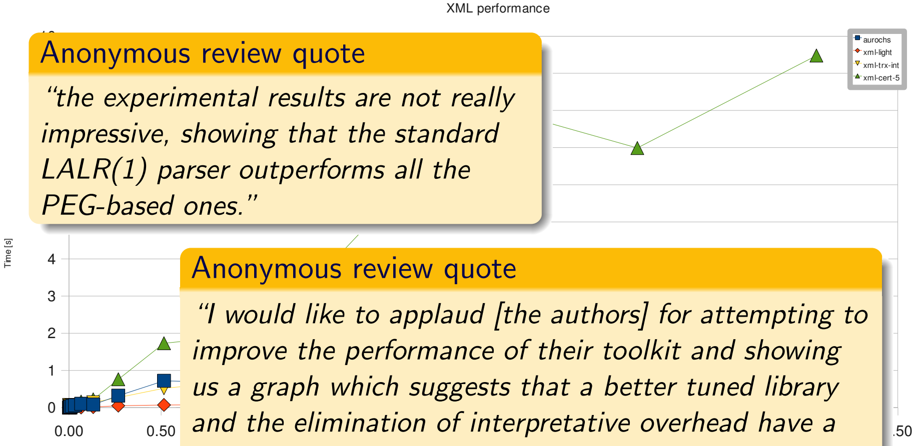
"the experimental results are not really impressive, showing that the standard LALR(1) parser outperforms all the PEG-based ones."

Anonymous review quote

"I would like to applaud [the authors] for attempting to improve the performance of their toolkit and showing us a graph which suggests that a better tuned library and the elimination of interpretative overhead have a good chance of delivering competitive results. The mere fact that it is even feasible to consider 'Performance comparison' is progress indeed."

- 7x slower
- 32x slowe

TRX: performance comparison with other tools



Anonymous review quote

"the experimental results are not really impressive, showing that the standard LALR(1) parser outperforms all the PEG-based ones."

Anonymous review quote

"I would like to applaud [the authors] for attempting to improve the performance of their toolkit and showing us a graph which suggests that a better tuned library and the elimination of interpretative overhead have a good chance of delivering competitive results. The mere fact that it is even feasible to consider 'Performance comparison' is progress indeed."

- 7x slower
- 32x slowe

Related work

Work on formally verified parsing:

- SLR parser in HOL

  Aditi Barthwal and Michael Norrish:
  Verified, Executable Parsing.
  ESOP '09

Related work

Work on formally verified parsing:

- SLR parser in HOL

  Aditi Barthwal and Michael Norrish:
  Verified, Executable Parsing.
  ESOP '09

- library of parser combinators in Agda

  Nils Anders Danielsson and Ulf Norell
  Structurally Recursive Descent Parsing.
  Draft, '08

Related work

Work on formally verified parsing:

- SLR parser in HOL

  Aditi Barthwal and Michael Norrish:
  Verified, Executable Parsing.
  ESOP '09

- library of parser combinators in Agda

  Nils Anders Danielsson and Ulf Norell
  Structurally Recursive Descent Parsing.
  Draft, '08

- PEG parser in Ynot/Coq

  Ryan Wisnesky and Gregory Malecha and Greg Morrisett
  Certified Web Services in Ynot.
  WWV '09

Related work

Work on formally verified parsing:

- SLR parser in HOL (termination not addressed)

  Aditi Barthwal and Michael Norrish:
  Verified, Executable Parsing.
  ESOP '09

- library of parser combinators in Agda (type-indices ruling out left-recursion)

  Nils Anders Danielsson and Ulf Norell
  Structurally Recursive Descent Parsing.
  Draft, '08

- PEG parser in Ynot/Coq (termination not addressed)

  Ryan Wisnesky and Gregory Malecha and Greg Morrisett
  Certified Web Services in Ynot.
  WWV '09

Conclusions

- We presented an interpreter for PEGs developed in Coq,
- along with the proofs of semantic preservation and termination ensuring total correctness of parsing.
- Using Coq's extraction capabilities this enables us to obtain formally correct parsers.

Conclusions

- We presented an interpreter for PEGs developed in Coq,
- along with the proofs of semantic preservation and termination ensuring total correctness of parsing.
- Using Coq's extraction capabilities this enables us to obtain formally correct parsers.

MLstate

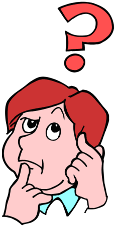
Conclusions

- We presented an interpreter for PEGs developed in Coq,
- along with the proofs of semantic preservation and termination ensuring total correctness of parsing.
- Using Coq's extraction capabilities this enables us to obtain formally correct parsers.

Conclusions

- We presented an interpreter for PEGs developed in Coq,
- along with the proofs of semantic preservation and termination ensuring total correctness of parsing.
- Using Coq's extraction capabilities this enables us to obtain formally correct parsers.

If you want to learn more about OPA I will be happy to discuss it over lunch!



http://mlstate.com

Thank you for your attention.