Présentation d'article:

CoStar: A Verified ALL(*) Parser

Par Mélissa Vallée

Sam Lasser
samuel.lasser@tufts.edu
Tufts University, USA

Chris Casinghino
ccasinghino@draper.com
Draper, USA

Kathleen Fisher
kfisher@cs.tufts.edu
Tufts University, USA

Cody Roux
croux@draper.com
Draper, USA

Mise en contexte

- Date : du 20 au 25 Juin 2021
- Auteurs: Sam Lasser, Chris Casinghino, Kathleen Fisher, Cody Roux
- Conférence: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '21)*
- Domaine: Logiciels et ingénierie logicielle → Analyseurs syntaxiques (Parsers) ; Vérification formelle des logiciels.

Problématique générale

Les analyseurs syntaxiques vérifiés existants présentent plusieurs limitations majeures :

- **Expressivité restreinte** → Compatibles avec un nombre limité de grammaires.
  Ex: les analyseurs descendants sont souvent limités à LL(1).
- **Problèmes de terminaison** → Certains analyseurs ne garantissent pas de toujours s'arrêter.
  Ex: les analyseurs ascendants peuvent boucler indéfiniment sur certaines entrées.
- **Manque de performance** → Inefficaces pour les langages et formats de données réels.
  Ex: trop lents ou gourmands en mémoire pour des grammaires complexes.

État de l'art

Plusieurs approches existent pour les analyseurs syntaxiques vérifiés, mais elles ont des limites.

- **Parsers Top-down LL(1)**

Avantages : Rapides et simples à implémenter.

Limite : Peuvent seulement traiter des grammaires sans ambiguïté ni récursivité gauche.

- **Parsers Bottom-up (ex: LR, LALR, GLR)**

Avantage : Compatibles avec un plus large éventail de grammaires.

Limite : Certains ne garantissent **pas toujours la terminaison** et peuvent être complexes à prouver formellement.

- **Parsers non vérifiés (ex: ANTLR, yacc, bison)**

Avantage : Performants, utilisés dans les compilateurs réels.

Limite : **Aucune garantie de correction formelle**, ce qui peut être problématique pour la sécurité des logiciels.

Problème central identifié :

- Il n’existe pas de parser vérifié qui soit à la fois expressif, performant et qui garantit la terminaison, ce qui pose un problème pour la conception de logiciels sécurisés.

CoStar

- **Correction et complétude** → CoStar assure que pour toute grammaire qui n'est pas left recursive, **un arbre syntaxique correct** est produit si l'entrée est valide.
- **Garanties de terminaison** → L'analyseur **s'arrête toujours sans erreur**, ce qui renforce sa fiabilité.
- **Détection de l'ambiguïté** → CoStar **identifie correctement** si une entrée peut être interprétée de plusieurs manières en marquant les arbres syntaxiques comme **uniques ou ambigus**.
- **Performance en temps linéaire** → Il **maintient des performances optimales** sur des grammaires non ambiguës utilisées dans les **langages de programmation et formats de données réels**.

Méthodologie

- Implémentation d’un analyseur syntaxique basé sur **ALL(*)**.
- Vérification formelle avec **Coq Proof Assistant**.
- Preuves de correction, terminaison et gestion de l’ambiguïté.
- Évaluation des performances sur des **grammaires réelles**.

Réalisation

API

- Le point d'entrée de CoStar est la fonction parse qui prend une grammaire G, un symbole start S ∈ N et un mot w
- Retourne une de ces valeurs :
  - Un arbre syntaxique **v** avec **S** comme racine et **w** comme feuilles. L'arbre est étiqueté **Unique** ou **Ambig**
  - Une valeur **Reject** indiquant que **w ∉ L(G)**.
  - Une valeur **Error** indiquant que l'analyseur a atteint un état incohérent.

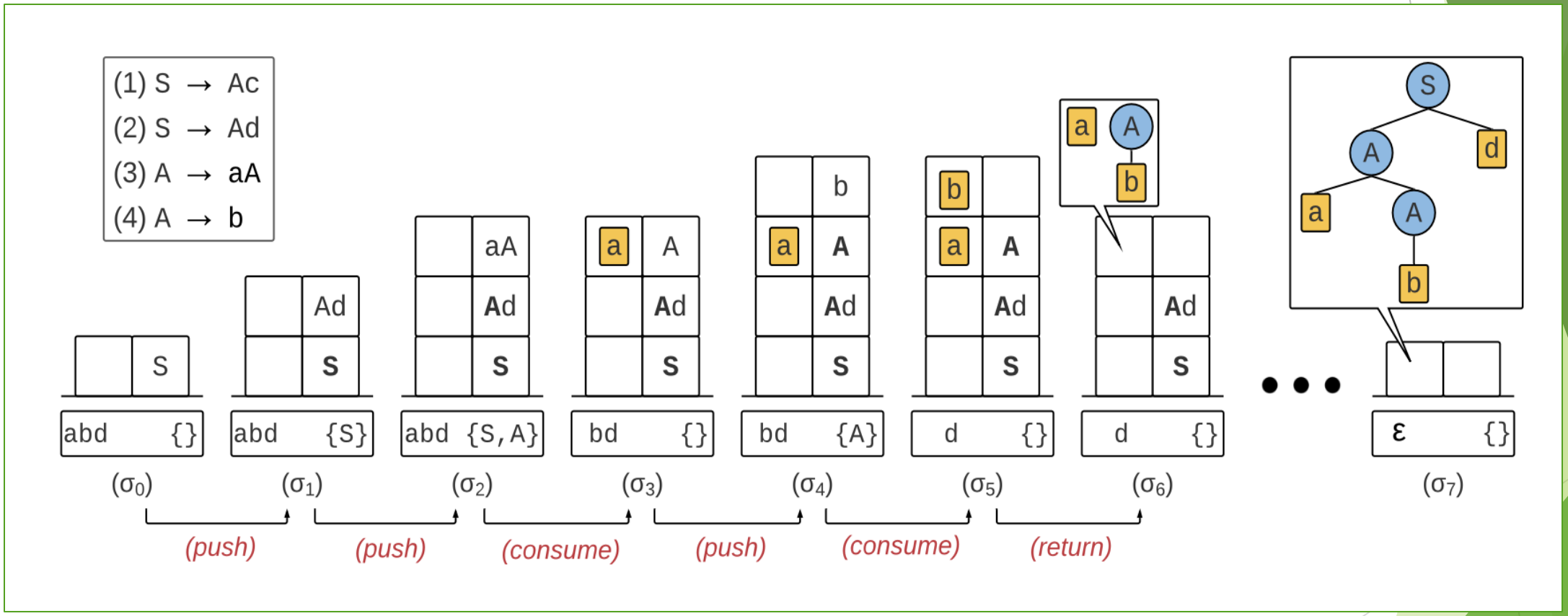
Composants de l'état de la machine

- Une pile **préfix**
- Une pile **suffix**
- Une **cache** pour stocker les résultats des prédictions précédantes.
- Un ensemble de non terminaux déjà visité.*
- Un flag unique (booléen) pour détecter si la syntaxe est ambigue

Fonctions principales

- **Push :** Quand le symbole au sommet de la pile suffix est un terminal → adaptivePredict
- **Consume :** Quand le symbole au sommet de la pile suffix est un non-terminal
- **Return :** Quand le sommet de la pile suffix est vide et la case en dessous(caller frame) contient un non-terminal ouvert X.

Exemple d'état des piles



- Flag unique à true tout le long.

AdaptivePredict de All(*)

- **CoStar** utilise **adaptivePredict** lorsque le sommet de la pile est un **non-terminal X** avec sa fonction push.
- La prédiction choisit quel **membre droit de la grammaire** pour **X** sera empilé sur la pile des suffixes.
- **adaptivePredict** combine deux stratégies de prédiction : **LL et SLL (Strong LL)**.
  - tente d'abord de faire une prédiction en mode **SLL**
  - bascule en mode **LL** uniquement lorsqu'il détecte que le résultat **SLL** peut être incorrect

Expérimentation

- Évaluer **CoStar** sur différentes grammaires (JSON, XML, DOT, Python 3)
- Réutilisé la grammaire XML, JSON et DOT de l'évaluation de performance original de ANTLR
- Ont exécuté les benchmarks sur un ordinateur portable avec 4 cœurs à 2,5 GHz, 7 Go de RAM et avec Ubuntu 16.04 comme OS
- Ont utilisé le compilateur OCaml version 4.11.1+flambda avec un niveau d'optimisation **-O3**
- Comparaison de performance avec ANTLR
- Test d'ambiguité

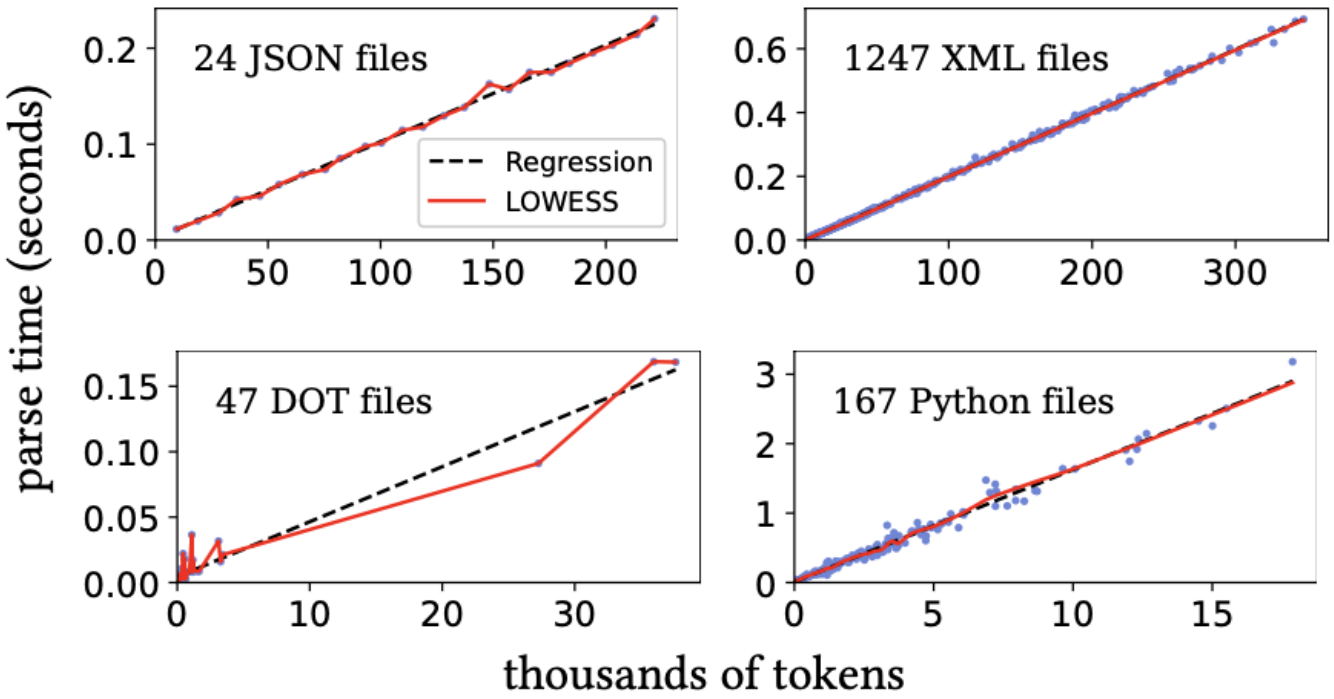
Benchmark	Grammar Size			Data Set Size	
	$\|\mathcal{T}\|$	$\|\mathcal{N}\|$	$\|\mathcal{P}\|$	# files	MB
JSON	11	7	17	25	21
XML	16	22	40	1260	192
DOT	20	44	73	48	19
Python 3	89	287	521	169	4

- Les mesures de la taille des grammaires et des ensembles de données pour les quatre benchmarks de CoStar
- Le compte de terminals T, non-terminal N et production P

Résultats

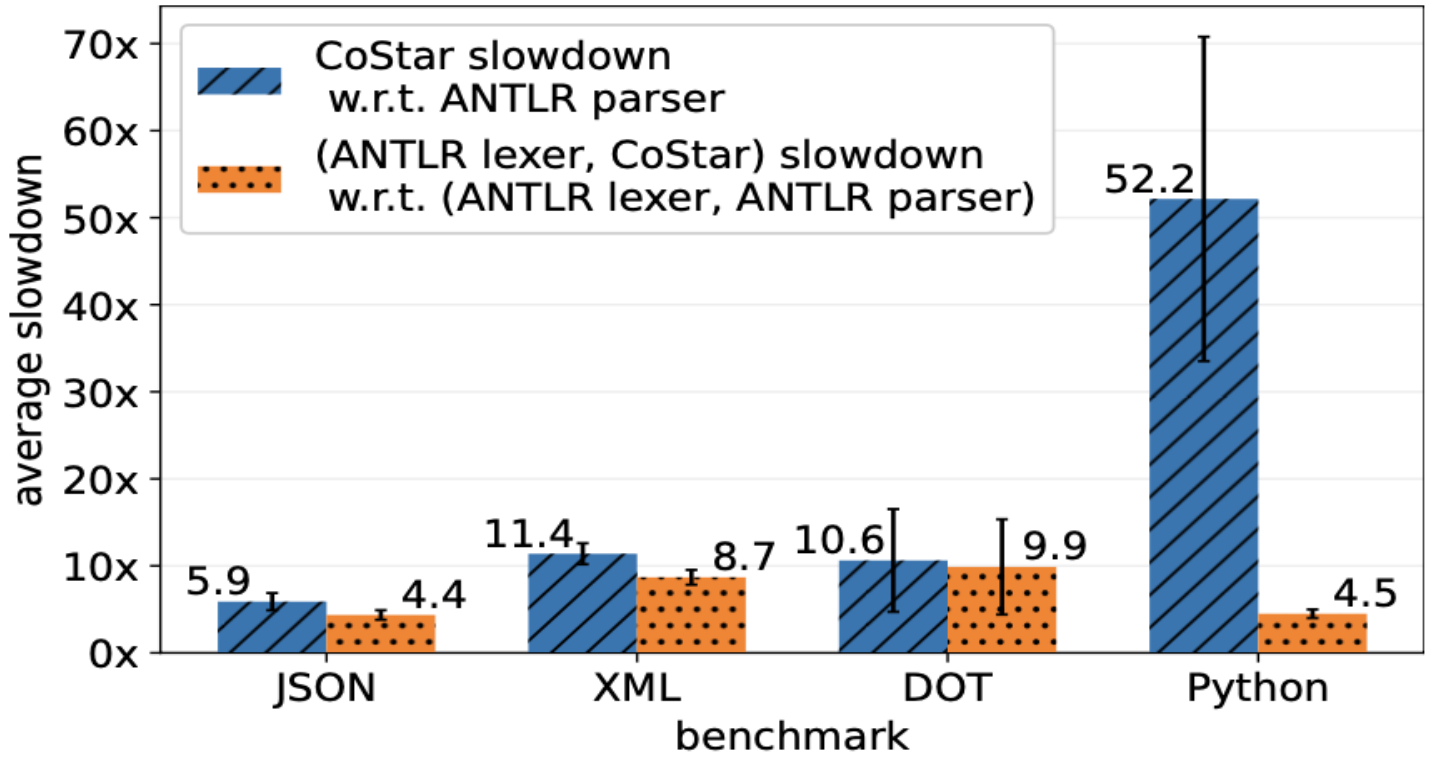
- CoStar génère des **arbres syntaxiques uniques** → confirmation de l'absence d'ambiguïté.
- CoStar **évite la récursion gauche** et **garantit la terminaison**.
- **Adaptabilité** : fonctionne efficacement sur **des grammaires variées et complexes.**
- **Expressivité d'ALL(*)** : Exemple en XML, prise en charge de règles avancées
- Ne retourne jamais d'erreur (à moin d'une recursion par la gauche)
- Complexité linéaire observé

Résultats



Chaque point représente le parse time de fichiers d'entrées individuels

Résultats


CoStar slowdown
w.r.t. ANTLR parser
(ANTLR lexer, CoStar) slowdown
w.r.t. (ANTLR lexer, ANTLR parser)
average slowdown
70x
60x
50x
40x
30x
20x
10x
0x
5.9
4.4
11.4
8.7
10.6
9.9
52.2
4.5
JSON
XML
DOT
Python
benchmark

Limites

- **CoStar ne supporte pas la récursivité à gauche.**
- **ANTLR contourne ce problème** en réécrivant les grammaires pour éliminer la récursion à gauche.
- **CoStar n'effectue pas ces réécritures de grammaire**, laissant cette tâche pour des travaux d'amélioration futurs.