Soutenance de stage
Alt-Ergo-Fuzz:
Un fuzzeur pour le solveur SMT Alt-Ergo

Hichem Rami Ait El Hara

OCamlPro - Sorbonne Université

8 septembre 2021

Sommaire

Préliminaires: les solveurs SAT

- SAT = satisfiabilité booléenne en logique propositionnelle
- un solveur SAT:
  - Cherche une instanciation des variables d'une formule logique telle que la formule soit vraie
  - Si il en trouve une, on dis que la formule est satisfiable (`sat`)
  - Sinon elle est insatisfiable (`unsat`)

Préliminaires: les solveurs SAT

- SAT = satisfiabilité booléenne en logique propositionnelle
- un solveur SAT:
  - Cherche une instanciation des variables d'une formule logique telle que la formule soit vraie
  - Si il en trouve une, on dis que la formule est satisfiable (`sat`)
  - Sinon elle est insatisfiable (`unsat`)

Préliminaires: les solveurs SAT

- SAT = satisfiabilité booléenne en logique propositionnelle
- un solveur SAT:
  - Cherche une instanciation des variables d'une formule logique telle que la formule soit vraie
  - Si il en trouve une, on dis que la formule est satisfiable (`sat`)
  - Sinon elle est insatisfiable (`unsat`)

Préliminaires: les solveurs SAT

- SAT = satisfiabilité booléenne en logique propositionnelle
- un solveur SAT:
  - Cherche une instanciation des variables d'une formule logique telle que la formule soit vraie
  - Si il en trouve une, on dis que la formule est satisfiable (`sat`)
  - Sinon elle est insatisfiable (`unsat`)

Préliminaires: les solveurs SAT

- SAT = satisfiabilité booléenne en logique propositionnelle
- un solveur SAT:
  - Cherche une instanciation des variables d'une formule logique telle que la formule soit vraie
  - Si il en trouve une, on dis que la formule est satisfiable (`sat`)
  - Sinon elle est insatisfiable (`unsat`)

Préliminaires: les solveurs SMT

Un solveur de Satisfiabilé Modulo Théories (SMT)

=

Solveur SAT + Procédures de décision pour les théories

Les solveurs SMT ont pour but de déterminer la satisfiabilité de formules définies dans une combinaison de théories en logique du premier ordre.

Certaines théories étant indécidables, les solveurs SMT peuvent aussi ne pas réussir à déterminer la satisfiabilité d'une formule, dans ce cas ils répondent avec `unknown`.

Grâce à leur expressivité assez riche, ils sont très utilisés pour la preuve de programmes.

Préliminaires: les solveurs SMT

Un solveur de Satisfiabilé Modulo Théories (SMT)

=

Solveur SAT + Procédures de décision pour les théories

Les solveurs SMT ont pour but de déterminer la satisfiabilité de formules définies dans une combinaison de théories en logique du premier ordre.

Certaines théories étant indécidables, les solveurs SMT peuvent aussi ne pas réussir à déterminer la satisfiabilité d'une formule, dans ce cas ils répondent avec `unknown`.

Grâce à leur expressivité assez riche, ils sont très utilisés pour la preuve de programmes.

Préliminaires: les solveurs SMT

Un solveur de Satisfiabilé Modulo Théories (SMT)

=

Solveur SAT + Procédures de décision pour les théories

Les solveurs SMT ont pour but de déterminer la satisfiabilité de formules définies dans une combinaison de théories en logique du premier ordre.

Certaines théories étant indécidables, les solveurs SMT peuvent aussi ne pas réussir à déterminer la satisfiabilité d'une formule, dans ce cas ils répondent avec `unknown`.

Grâce à leur expressivité assez riche, ils sont très utilisés pour la preuve de programmes.

Préliminaires: les solveurs SMT

Un solveur de Satisfiabilé Modulo Théories (SMT)

=

Solveur SAT + Procédures de décision pour les théories

Les solveurs SMT ont pour but de déterminer la satisfiabilité de formules définies dans une combinaison de théories en logique du premier ordre.

Certaines théories étant indécidables, les solveurs SMT peuvent aussi ne pas réussir à déterminer la satisfiabilité d'une formule, dans ce cas ils répondent avec `unknown`.

Grâce à leur expressivité assez riche, ils sont très utilisés pour la preuve de programmes.

Motivation

- Les logiciels sont partout
  - On leur confie des tâches plus compliquées et plus critiques qu'avant.
  - La présence de bugs peut être très dangereuse.

- Les solveurs SMT:
  - Font partie des outils qui évitent la présence de bugs.
  - Plusieurs outils de vérification de programmes en dépendent.
  - Ils sont eux-mêmes des logiciels complexes donc ...

Motivation

- Les logiciels sont partout
  - On leur confie des tâches plus compliquées et plus critiques qu'avant.
  - La présence de bugs peut être très dangereuse.
- Les solveurs SMT:
  - Font partie des outils qui évitent la présence de bugs.
  - Plusieurs outils de vérification de programmes en dépendent.
  - Ils sont eux-mêmes des logiciels complexes donc ...

Le solveur SMT Alt-Ergo

Alt-Ergo est un solveur SMT:

- Développé en 2006 au LRI, maintenu par OCamlPro depuis 2013
- Programmé en OCaml.
- Open source (https://github.com/ocamlpro/alt-ergo).
- Conçu pour la preuve de programmes.
- Utilisé comme backend par: Why3, Frama-C, SPARK ...

Le solveur SMT Alt-Ergo

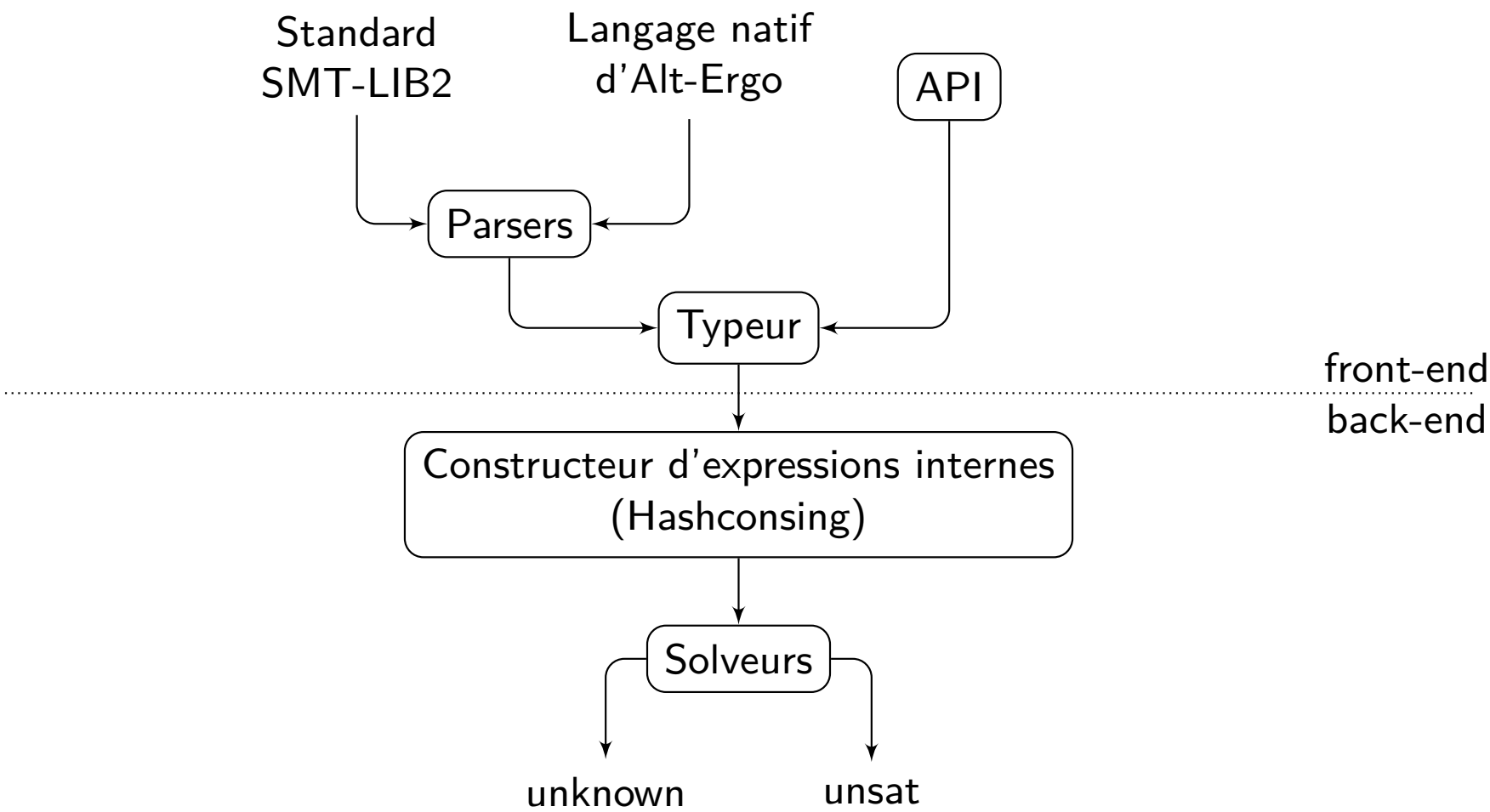

Figure: Architecture d'Alt-Ergo

Le fuzzing avec AFL

AFL est un fuzzeur (https://github.com/google/AFL) qui utilise:

- Une instrumentation de code
- Des algorithmes génétiques

AFL a déjà servi à trouver des bugs dans plusieurs logiciels (cf. la "bug-o-rama" sur `https://lcamtuf.coredump.cx/afl/`)

Le fuzzing avec AFL

AFL est un fuzzeur (https://github.com/google/AFL) qui utilise:

- Une instrumentation de code
- Des algorithmes génétiques

AFL a déjà servi à trouver des bugs dans plusieurs logiciels (cf. la "bug-o-rama" sur `https://lcamtuf.coredump.cx/afl/`)

Le fuzzing avec AFL

AFL est un fuzzeur (https://github.com/google/AFL) qui utilise:

- Une instrumentation de code
- Des algorithmes génétiques

AFL a déjà servi à trouver des bugs dans plusieurs logiciels (cf. la "bug-o-rama" sur `https://lcamtuf.coredump.cx/afl/`)

Le fuzzing avec AFL

AFL est un fuzzeur (https://github.com/google/AFL) qui utilise:

- Une instrumentation de code
- Des algorithmes génétiques

AFL a déjà servi à trouver des bugs dans plusieurs logiciels (cf. la "bug-o-rama" sur `https://lcamtuf.coredump.cx/afl/`)

Le fuzzing avec AFL

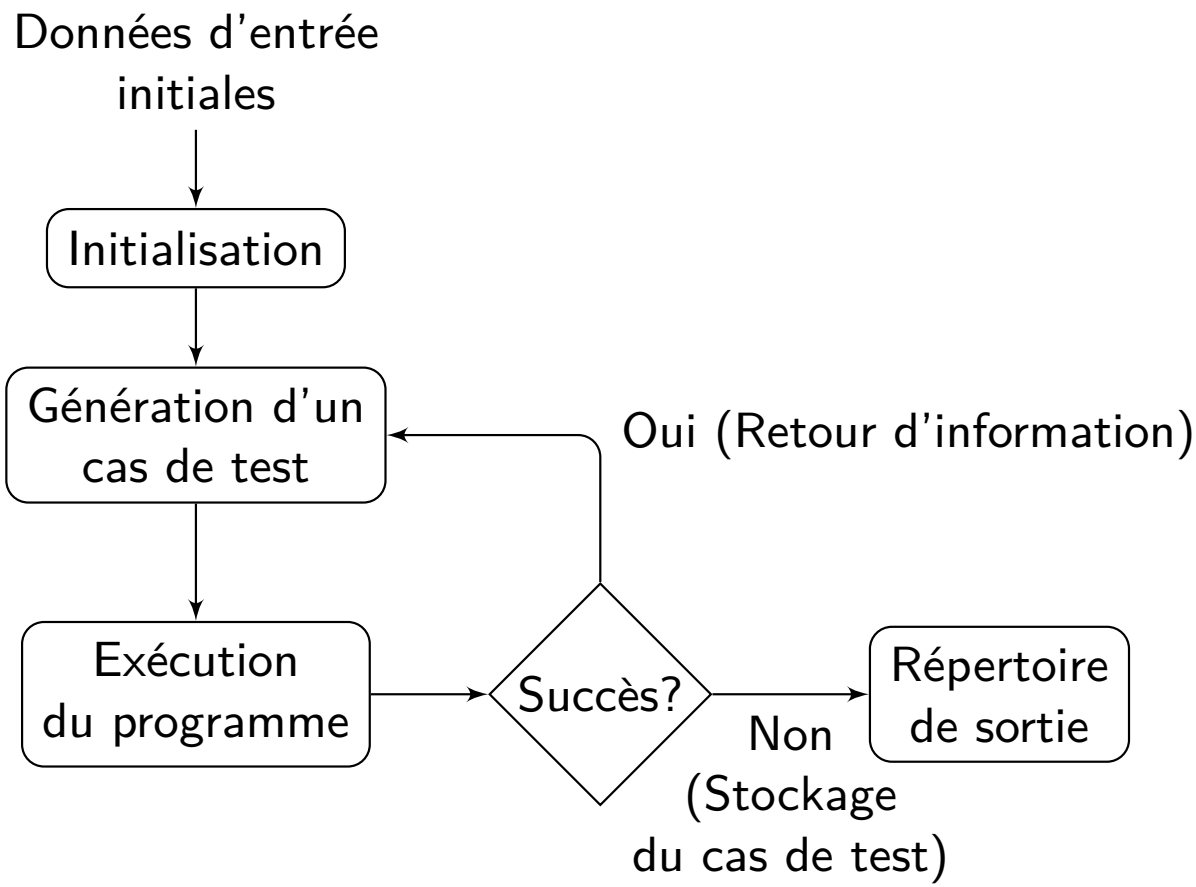

Figure: Fonctionnement d'un fuzzeur

A ce stade:

- On connaît Alt-Ergo
- On connaît AFL

Comment lier les deux?
À travers Crowbar

A ce stade:

- On connaît Alt-Ergo
- On connaît AFL

Comment lier les deux?

À travers Crowbar

A ce stade:

- On connaît Alt-Ergo
- On connaît AFL

Comment lier les deux?
À travers Crowbar

La librarie Crowbar

Crowbar (github.com/stedolan/crowbar) est une libraire OCaml qui:

- Fournit des générateurs pour les types de base d'OCaml, ainsi que des combinateurs de générateurs.
- Permet de combiner des tests à la QuickCheck avec le fuzzing.

Avec cette librairie nous avons ce qu'il nous faut pour construire le fuzzeur.

La librarie Crowbar

Crowbar (github.com/stedolan/crowbar) est une libraire OCaml qui:

- Fournit des générateurs pour les types de base d'OCaml, ainsi que des combinateurs de générateurs.
- Permet de combiner des tests à la QuickCheck avec le fuzzing.

Avec cette librairie nous avons ce qu'il nous faut pour construire le fuzzeur.

La librarie Crowbar

Crowbar (github.com/stedolan/crowbar) est une libraire OCaml qui:

- Fournit des générateurs pour les types de base d'OCaml, ainsi que des combinateurs de générateurs.
- Permet de combiner des tests à la QuickCheck avec le fuzzing.

Avec cette librairie nous avons ce qu'il nous faut pour construire le fuzzeur.

La librarie Crowbar

Crowbar (github.com/stedolan/crowbar) est une libraire OCaml qui:

- Fournit des générateurs pour les types de base d'OCaml, ainsi que des combinateurs de générateurs.
- Permet de combiner des tests à la QuickCheck avec le fuzzing.

Avec cette librairie nous avons ce qu'il nous faut pour construire le fuzzeur.

AST (Abstract Syntax Tree)

Le langage intermédiaire supporte les types suivants:

- `int` : Les entiers;
- `real` : Les rationnels;
- `bool` : Les booléens;
- `bitvec n` : Les vecteurs de bits de taille n;
- `(a, b) farray` : Les tableaux fonctionnels;
- `adt (string, (string, (string, t) list) list)` : Les types de données algébriques.

AST (Abstract Syntax Tree)

Et les déclarations SMT suivantes:

- `Goal` : Un but à prouver par le solveur
- `Axiom` : Un axiome utilisé par le solveur afin de l'aider à prouver des buts
- `Predicate` : Un prédicat dans le sens logique du terme, une fonction qui prend des arguments et retourne une valeur booléenne
- `Function` : Une fonction qui prend des arguments et retourne une valeur non booléenne

AST (Abstract Syntax Tree)

Ainsi que les structures de contrôle suivantes:

- `let [...] in [...];`
- `If [...] then [...] else [...];`
- `match [...] with [...].`

Et les expressions suivante:

- `const` : une constante qui est une des deux:
  - les littéraux
  - les symboles non-interprétés.
- `var` : une variable de n'importe quel type et d'une de ces catégories:
  - les variables quantifiées;
  - les variables liées;
- `app` : une application qui est soit un appel de fonction, soit une application d'un opérateur.

Génération de formules SMT

Nous ne cherchons pas à fuzzeur les parseurs ni le typeur d'Alt-Ergo, c'est les solveurs qui nous intéresse.

Pour cela nous souhaitons générer des déclarations SMT bien formées.

Ce qui introduit des invariants que les formules générées doivent respecter:

- Pas d'expressions mal-typées;
- Toute variable liée doit être couverte par sa liaison;
- Toute instance d'une variable quantifiées doit l'être aussi;
- Pas de fonction ou de variable non-définie.

Génération de formules SMT

Nous ne cherchons pas à fuzzeur les parseurs ni le typeur d'Alt-Ergo, c'est les solveurs qui nous intéresse.

Pour cela nous souhaitons générer des déclarations SMT bien formées.

Ce qui introduit des invariants que les formules générées doivent respecter:

- Pas d'expressions mal-typées;
- Toute variable liée doit être couverte par sa liaison;
- Toute instance d'une variable quantifiées doit l'être aussi;
- Pas de fonction ou de variable non-définie.

Génération de formules SMT

Nous ne cherchons pas à fuzzeur les parseurs ni le typeur d'Alt-Ergo, c'est les solveurs qui nous intéresse.

Pour cela nous souhaitons générer des déclarations SMT bien formées.

Ce qui introduit des invariants que les formules générées doivent respecter:

- Pas d'expressions mal-typées;
- Toute variable liée doit être couverte par sa liaison;
- Toute instance d'une variable quantifiées doit l'être aussi;
- Pas de fonction ou de variable non-définie.

Génération de formules SMT

Nous ne cherchons pas à fuzzeur les parseurs ni le typeur d'Alt-Ergo, c'est les solveurs qui nous intéresse.

Pour cela nous souhaitons générer des déclarations SMT bien formées.

Ce qui introduit des invariants que les formules générées doivent respecter:

- Pas d'expressions mal-typées;
- Toute variable liée doit être couverte par sa liaison;
- Toute instance d'une variable quantifiées doit l'être aussi;
- Pas de fonction ou de variable non-définie.

Génération de formules SMT

Nous ne cherchons pas à fuzzeur les parseurs ni le typeur d'Alt-Ergo, c'est les solveurs qui nous intéresse.

Pour cela nous souhaitons générer des déclarations SMT bien formées.

Ce qui introduit des invariants que les formules générées doivent respecter:

- Pas d'expressions mal-typées;
- Toute variable liée doit être couverte par sa liaison;
- Toute instance d'une variable quantifiées doit l'être aussi;
- Pas de fonction ou de variable non-définie.

Traduction vers les langages SMT

Pour qu'elles puissent être résolues nous offrons la possibilité de les traduire vers:

- Le langage interne d'Alt-Ergo
- Le langage natif d'Alt-Ergo
- Le standard SMT-LIB2

Traduction vers les langages SMT

Pour qu'elles puissent être résolues nous offrons la possibilité de les traduire vers:

- Le langage interne d'Alt-Ergo
- Le langage natif d'Alt-Ergo
- Le standard SMT-LIB2

Traduction vers les langages SMT

Pour qu'elles puissent être résolues nous offrons la possibilité de les traduire vers:

- Le langage interne d'Alt-Ergo
- Le langage natif d'Alt-Ergo
- Le standard SMT-LIB2

Traduction vers les langages SMT

Pour qu'elles puissent être résolues nous offrons la possibilité de les traduire vers:

- Le langage interne d'Alt-Ergo
- Le langage natif d'Alt-Ergo
- Le standard SMT-LIB2

A ce stade nous connaissons:

- Le solveur SMT Alt-Ergo
- Le fuzzeur AFL
- La librarie Crowbar

Et de ce que nous avons créé:

- Le langage intermédiaire
- La génération de formules SMT
- La traduction vers les langages SMT

Quelle propriétés souhaitons nous vérifier?
Et quels genre bugs cherchons nous a trouver?

A ce stade nous connaissons:

- Le solveur SMT Alt-Ergo
- Le fuzzeur AFL
- La librarie Crowbar

Et de ce que nous avons créé:

- Le langage intermédiaire
- La génération de formules SMT
- La traduction vers les langages SMT

Quelle propriétés souhaitons nous vérifier?
Et quels genre bugs cherchons nous a trouver?

Figure Pseudocode de la fonction de test

```
1:  function (stmts: list of SMT statements)
2:      try
3:          ae_stmts ← translate_ae(stmts)
4:          smt2_stmts ← translate_smt2(stmts)
5:          cvc5_res ← cvc5_t_solve(smt2_stmts)
6:          ae_t_res ← ae_t_solve(ae_stmts)     ▷ Using the Tableaux solver
7:          ae_c_res ← ae_c_solve(ae_stmts)       ▷ Using the CDCL solver
8:          try
9:              cmp_answers_exn3(cvc5_res, ae_t_res, ae_c_res)
10:         catch exn
11:             handle_bug(exn, stmts, cvc5_res, ae_t_res, ae_c_res)
12:         end try
13:     catch exn
14:         handle_bug_na(exn, stmts)
15:     end try
16: end function
```

Gestion et reproduction des bugs

La gestion des bugs consiste en:

- Le marshalling d'une structure de données contenant des informations sur le bug.
- L'écriture des données marshallées dans un fichier.

Le reproduction des bugs peut se faire:

- En traduisant le contenu d'un fichier marshallé vers un fichier ".smt2" et/ou ".ae" et en relançant le solveur dessus.
- Ou en relançant directement la réexécution de la fonction de test qui répondra sur la sortie standard.

Gestion et reproduction des bugs

La gestion des bugs consiste en:

- Le marshalling d'une structure de données contenant des informations sur le bug.
- L'écriture des données marshallées dans un fichier.

Le reproduction des bugs peut se faire:

- En traduisant le contenu d'un fichier marshallé vers un fichier ".smt2" et/ou ".ae" et en relançant le solveur dessus.
- Ou en relançant directement la réexécution de la fonction de test qui répondra sur la sortie standard.

Gestion et reproduction des bugs

La gestion des bugs consiste en:

- Le marshalling d'une structure de données contenant des informations sur le bug.
- L'écriture des données marshallées dans un fichier.

Le reproduction des bugs peut se faire:

- En traduisant le contenu d'un fichier marshallé vers un fichier ".smt2" et/ou ".ae" et en relançant le solveur dessus.
- Ou en relançant directement la réexécution de la fonction de test qui répondra sur la sortie standard.

Gestion et reproduction des bugs

La gestion des bugs consiste en:

- Le marshalling d'une structure de données contenant des informations sur le bug.
- L'écriture des données marshallées dans un fichier.

Le reproduction des bugs peut se faire:

- En traduisant le contenu d'un fichier marshallé vers un fichier ".smt2" et/ou ".ae" et en relançant le solveur dessus.
- Ou en relançant directement la réexécution de la fonction de test qui répondra sur la sortie standard.

Gestion et reproduction des bugs

La gestion des bugs consiste en:

- Le marshalling d'une structure de données contenant des informations sur le bug.
- L'écriture des données marshallées dans un fichier.

Le reproduction des bugs peut se faire:

- En traduisant le contenu d'un fichier marshallé vers un fichier ".smt2" et/ou ".ae" et en relançant le solveur dessus.
- Ou en relançant directement la réexécution de la fonction de test qui répondra sur la sortie standard.

Gestion et reproduction des bugs

La gestion des bugs consiste en:

- Le marshalling d'une structure de données contenant des informations sur le bug.
- L'écriture des données marshallées dans un fichier.

Le reproduction des bugs peut se faire:

- En traduisant le contenu d'un fichier marshallé vers un fichier ".smt2" et/ou ".ae" et en relançant le solveur dessus.
- Ou en relançant directement la réexécution de la fonction de test qui répondra sur la sortie standard.

Architecture d'Alt-Ergo-Fuzz

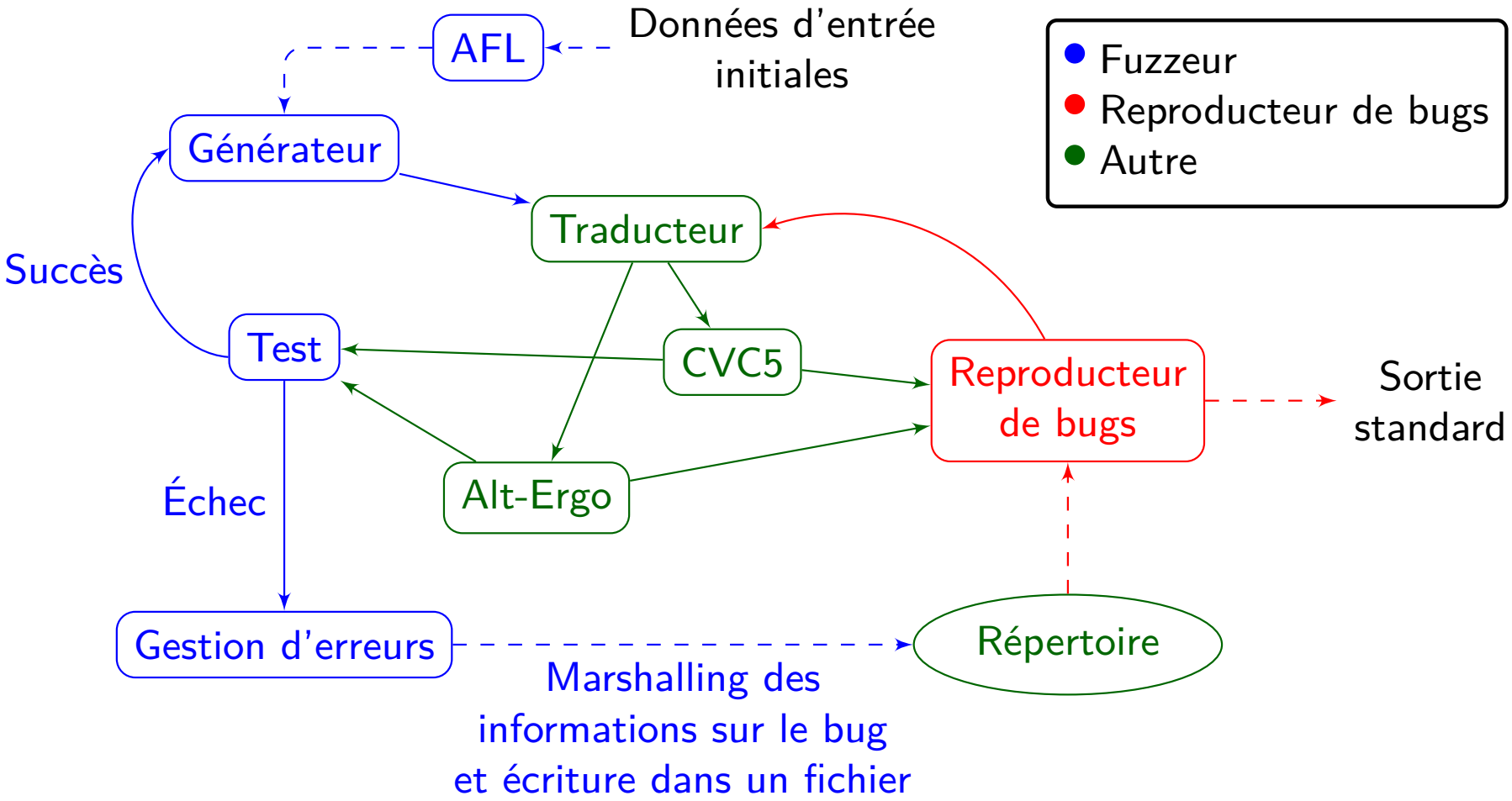

Figure: Architecture d'Alt-Ergo-Fuzz

Résultats

Après avoir été lancé en mode parallèle sur une machine à 4 coeurs ayant 8gb de ram pendant une durée de 20 jours, Alt-Ergo-Fuzz a trouvé:

7 bugs dont 4 crashes et 3 bugs d'unsoundness. Issues GitHub créées:

Unsoundness:

1. **issue 476:** github.com/OCamlPro/alt-ergo/issues/476
2. **issue 477:** github.com/OCamlPro/alt-ergo/issues/477
3. **issue 479:** github.com/OCamlPro/alt-ergo/issues/479

Crashes:

1. **issue 474:** github.com/OCamlPro/alt-ergo/issues/474
2. **issue 475:** github.com/OCamlPro/alt-ergo/issues/475
3. **issue 481:** github.com/OCamlPro/alt-ergo/issues/481
4. **issue 482:** github.com/OCamlPro/alt-ergo/issues/482

Limitations

- La sélection des paramètres de génération doit être faite d'une meilleure façon.
- La duplication des bugs, si on veut trouver des nouveaux bugs on doit résoudre ceux qu'on a déjà.
- La stabilité, après quelques heures d'exécution elle finit toujours par baisser. Il faut trouver pourquoi et résoudre le problème.

Limitations

- La sélection des paramètres de génération doit être faite d'une meilleure façon.
- La duplication des bugs, si on veut trouver des nouveaux bugs on doit résoudre ceux qu'on a déjà.
- La stabilité, après quelques heures d'exécution elle finit toujours par baisser. Il faut trouver pourquoi et résoudre le problème.

Limitations

- La sélection des paramètres de génération doit être faite d'une meilleure façon.
- La duplication des bugs, si on veut trouver des nouveaux bugs on doit résoudre ceux qu'on a déjà.
- La stabilité, après quelques heures d'exécution elle finit toujours par baisser. Il faut trouver pourquoi et résoudre le problème.

Conclusion

Pour récapituler:

- Durant cet exposé je vous ai présenté Alt-Ergo-Fuzz.
- Et les résultats de sa mise en pratique.
- Ainsi que les limitations que j'ai remarqué.

L'ensemble du code source d'Alt-Ergo-Fuzz est disponible à:

`https://github.com/hichaeh/alt-ergo-fuzz`

Conclusion

Pour récapituler:

- Durant cet exposé je vous ai présenté Alt-Ergo-Fuzz.
- Et les résultats de sa mise en pratique.
- Ainsi que les limitations que j'ai remarqué.

L'ensemble du code source d'Alt-Ergo-Fuzz est disponible à:

`https://github.com/hichaeh/alt-ergo-fuzz`

Conclusion

Pour récapituler:

- Durant cet exposé je vous ai présenté Alt-Ergo-Fuzz.
- Et les résultats de sa mise en pratique.
- Ainsi que les limitations que j'ai remarqué.

L'ensemble du code source d'Alt-Ergo-Fuzz est disponible à:

https://github.com/hichaeh/alt-ergo-fuzz

Conclusion

Pour récapituler:

- Durant cet exposé je vous ai présenté Alt-Ergo-Fuzz.
- Et les résultats de sa mise en pratique.
- Ainsi que les limitations que j'ai remarqué.

L'ensemble du code source d'Alt-Ergo-Fuzz est disponible à:

https://github.com/hichaeh/alt-ergo-fuzz

Conclusion

Pour récapituler:

- Durant cet exposé je vous ai présenté Alt-Ergo-Fuzz.
- Et les résultats de sa mise en pratique.
- Ainsi que les limitations que j'ai remarqué.

L'ensemble du code source d'Alt-Ergo-Fuzz est disponible à:

https://github.com/hichaeh/alt-ergo-fuzz

Merci pour votre attention!