SARDANA : An Abstract Interpretation Based Tool for Optimization of Numerical Expressions in LUSTRE Programs

Arnault Ioualalen, Matthieu Martel
University of Perpignan Via Domitia

TAPAS'10 - 17 september

ubo
université de bretagne occidentale

SARDANA

Ens

Fondation
de Recherche
pour l' Aéronautique
& l' Espace

UPVD
Université de Perpignan Via Domitia

Objectives

Sardana is a compiler

- for synchronous language like LUSTRE (critical system)
- which improves numerical accuracy by re-writing expressions.

Why talk about numerical accuracy ?

Float operators are not the same as the $\mathbb{R}$ operators $\Rightarrow$ rounding error !

What do we expect ?

Find an expression over $\mathbb{F}$ which the evaluation is a *good* approximation of the evaluation of the expression over $\mathbb{R}$.

Rounding Error Semantic (M. Martel, JHOSC'06)

Each variable $f \in \mathbb{R}$ is decomposed as

- A floating point number : $\uparrow_\circ (f) \in \mathbb{F}$
- A real rounding error : $\downarrow_\circ (f) \in \mathbb{R}$

such as $f = \uparrow_\circ (f) + \downarrow_\circ (f)$

Any operator is defined like the following

Ex : $(x_1, \epsilon_1) + (x_2, \epsilon_2) = (\uparrow_\circ (x_1 + x_2), \downarrow_\circ (x_1 + x_2) + \epsilon_1 + \epsilon_2)$

Current state of our analyzer

Current specification

- Works on simple LUSTRE langage (Caspi & al. POPL'87)
- Optimizes floating point numerical expressions composed with +, -, * operators

Is re-writing hard ?

Yes, usual laws of $\mathbb{R}$ field imply combinatorial explosion of number of equivalent expressions.
Also, there are very few *good* ways of writing a non-trivial expression (L.Thévenoux, P. Langlois, M.Martel, PASCO'10)

How to represent all these expressions ?

They share some parts. . .

. . . or at least they are derived one from an other through a re-writing rule.

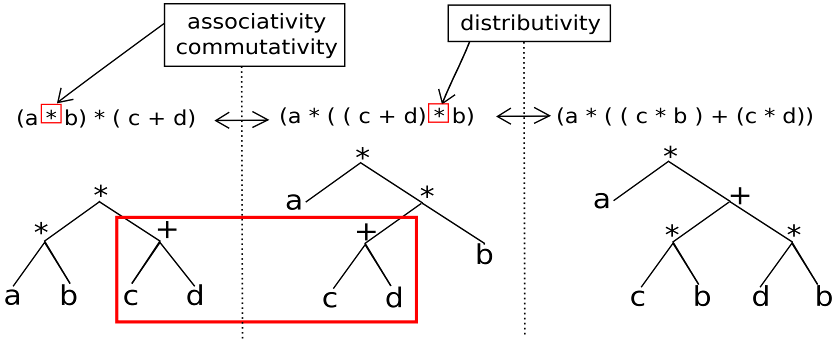


Figure: Example of relation between expressions

PEG and EPEGs, POPL'09, (R.Tate, M.Stepp, Z.Tatlock, S.Lerner)

Equivalence **P**rogram **E**xpansion **G**raph

Designed for the *phase ordering* problem.

PEGs and E-PEGs have properties of interest

- Useful to eliminate redundant information (compactness) for imperative program
- Build through a saturation of equivalence rules of program transformation (equivalence class)

But that's not enough

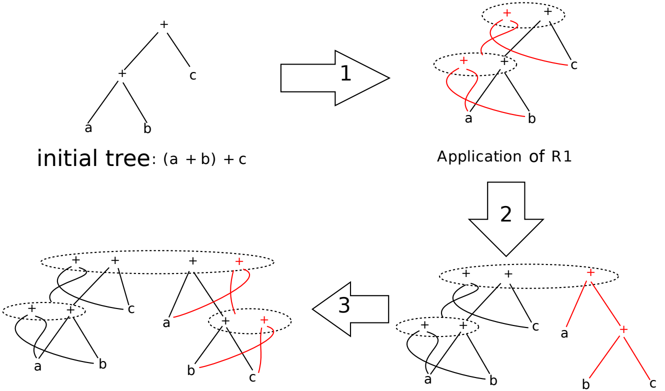
Even without redundancy, combinatory explosion doesn't fit in it.

E-PEG construction

An E-PEG is build as a transitive closure

The key point is the rules set used to make the saturation of the structure.

R1 : a + b = b + a
R2 : a + (b + c) = (a + b) + c



initial tree: (a + b) + c

Application of R1

Application of R2

Application of R1, ect...

Abstract EPEGs

We construct them directly from the syntactic tree

- Polynomial size
- Covers all the *equivalent* expressions
- Does NOT cover non-equivalent expressions

The expression equivalency relation is defined by the usual rules :

- Associativity
- Commutativity
- Distributivity
- Factorization

An abstraction box is defined by its parameters and an operator

- Whole sub-space covering of possible expressions
- Recursive structure
- Easy to infer the worst rounding error of the set

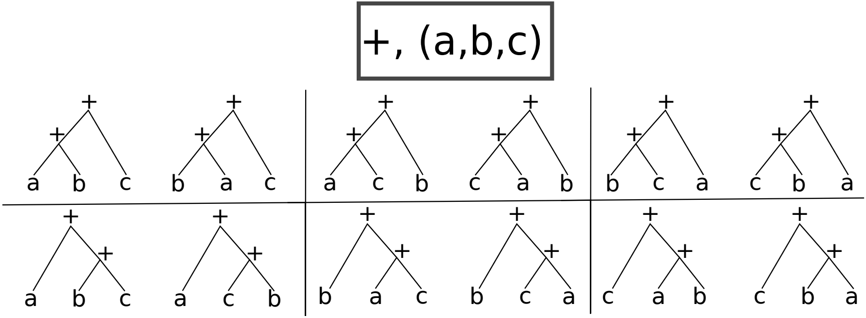


Figure: An abstract box containing 3 leaves

Abstractions (Work in progress)

Main goal of the abstractions

Enhance the structure with abstraction boxes in order to cover all equivalent expressions.

3 abstractions are developed today

- Left-Right Abstraction
- Transverse Abstraction
- Box Expansion

All are integrated in our tool despite the non-completness

Non-trivial results could already been reached, helping us moving forward and find the other abstractions we need to cover the remaining expressions.

Left-Right Abstraction is heading down

Each node try to use the homogeneous structure of it's left sub-tree and right sub-tree.

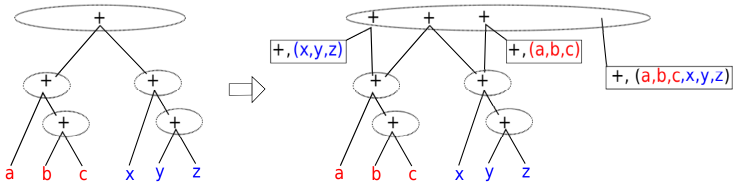


Figure: Illustration of the Left-Right Abstraction

The Abstraction Box at the right is called a Global Box, it stands for the less accurate abstraction.

Transverse Abstraction is heading up

In an homogeneous structure, we iteratively abstract the higher part.

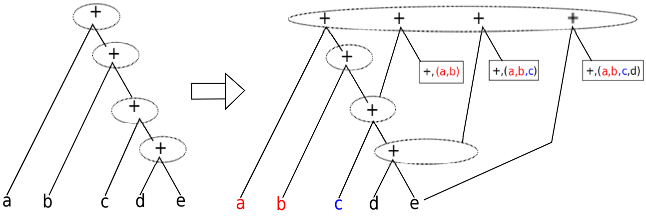


Figure: Illustration of the Transverse Abstraction

We also add boxes to link the leaves : a, b, c and d.

Box Expansion

This abstraction fuses abstract boxes when possible

When one of the parameters of a box is itself a box, and there is the same operator in both

$$\boxed{+,\ a,\ b,\ \boxed{+,\ x\ ,\ y}} \longrightarrow \boxed{+,\ a,\ b,\ x,\ y}$$

Figure: Illustration of the Box Expansion

What are the expressions we miss ?

Partial distributivity

Distribute all partial factors creates an exponential number of expressions.

Ex : $a * b * c * (x + y) =$

- $a * b * (c * x + c * y)$
- $a * c * (b * x + b * y)$
- ...
- $a * (b * c * x + b * c * y)$
- $b * (a * c * x + a * c * y)$
- ...

Currently we always distribute all the factors $\Rightarrow$ structure becomes homogeneous.

Symmetrically we have the same problem for partial factorization

Currently we always factorize by all the factors $\Rightarrow$ structure becomes homogeneous.

The extraction of a more precise program

We use a local heuristic

In each equivalence class we select the best candidate using the error rounding semantic

Currently we use the Max partial order over the stream of intervals

Other partial orders are going to be implemented like : Strict inclusion, or value of the integral.

Conclusion & Perspectives

Several achievement have been reached so far :

- Polynomial structure
- Complete loop of optimization from source to code to source code
- Graphical interface embedding completely the analyzer

Problems we are going to solve

- Analysis is not yet complete
- Analysis works on each variable separately
- Analysis doesn't take into account the recursive definition of variable
- Analysis has to work also onto any kind operators
- Tradeoff between time consumption and accuracy

The end

Thank you for your attention

Questions ?