From Trace Semantics for Imperative Programs to Regular and Context-Free Programs

Second Bachelor Seminar Talk

Clara Schneidewind
Advisor: Prof. Dr. Gert Smolka

SAARLAND UNIVERSITY
COMPUTER SCIENCE

July 17th, 2015

Content

Overview

- Initial motivation: A verified compiler from Imp to LImp
- Extension: Translating programs with I/O
- New semantics for verification: Trace semantics
- Lifting of Imp and LImp programs in a more general setting:

$$\begin{array}{cccc}
\text{Imp} & & & \\
& \subsetneq & & \\
& & \text{Regular Programs} \quad \subseteq \quad \text{Context-Free Programs} & \\
& \subsetneq & & \\
\text{LImp} & & &
\end{array}$$

Previous Work

Michael J. Fischer and Richard E. Landner
Propositional Dynamic Logic of Regular Programs
Journal of Computer and System Sciences, 1979

Dexter Kozen and Frederick Smith
Kleene Algebra with Tests: Completeness and Decidability
Springer, 1996

Joost Winter, Marcello M. Bonsangue, and Jan Rutten
Context-Free Languages, Coalgebraically
Springer, 2011

Karl R. Abrahamson
Succinct Representation of Regular Sets Using Gotos and Boolean Variables
Journal of Computer and System Sciences, 1987

Paul Morris, Ronald A. Gray and Robert E. Filman
GOTO Removal Based On Regular Expressions
Journal of Software Maintenance Research and Practice, 2000

Imp

Short review: Imp

Example

$$(\textbf{if } x < 0 \textbf{ then } x ::= -x \textbf{ else SKIP}) \; ; \; r ::= x$$

$$c, d \;\; ::= \;\; a \;\mid\; c \; ; \; d \;\mid\; \textbf{if } b \textbf{ then } c \textbf{ else } d \;\mid\; \textbf{while } b \textbf{ do } c$$

Example

$$s := (\textbf{if } b_{pos} \textbf{ then } a_{inv} \textbf{ else } a_{skip}) \; ; \; a_{res}$$

$$[x \mapsto -1], s \longrightarrow [x \mapsto -1], a_{inv} ; a_{res} \longrightarrow [x \mapsto 1], a_{res}$$

Stack Semantics

Example

$$s := (\textbf{if } b_{pos} \textbf{ then } a_{inv} \textbf{ else } a_{skip}) \; ; \; a_{res}$$

$$\begin{array}{rr}
[x \mapsto -1] & [(\textbf{if } b_{pos} \textbf{ then } a_{inv} \textbf{ else } a_{skip}) \; ; \; a_{res}], \\
\longrightarrow [x \mapsto -1] & [\textbf{if } b_{pos} \textbf{ then } a_{inv} \textbf{ else } a_{skip}, \; a_{res}] \\
\longrightarrow [x \mapsto -1] & [a_{inv}, \; a_{res}] \\
\longrightarrow [x \mapsto 1] & [a_{res}] \\
\longrightarrow [x \mapsto 1, r \mapsto 1] & []
\end{array}$$

- Decomposition of sequences on the stack
- Empty stack as indicator for termination

Observable Actions

Example

$$s := \quad \text{IN } x; \quad \left.\begin{array}{l} \textbf{if } (b_{pos}) \textbf{ then } a_{inv} \textbf{ else } a_{skip}; \\ \text{OUT } x \end{array}\right\} \mathit{if}$$

$$[], [s] \rightarrow [], [\text{IN } x,\ \text{if}] \xrightarrow{?\ \text{-}1} \cdots \rightarrow [x \mapsto 1], [\text{OUT } x] \xrightarrow{!\ 1} [x \mapsto 1], []$$

Observable actions require to track traces in addition to final state

Resulting traces: $\{\ldots,$ -2.2, -1.1, 0.0, 1.1, $\ldots\}$

Non-Termination

Reactive system:

Example

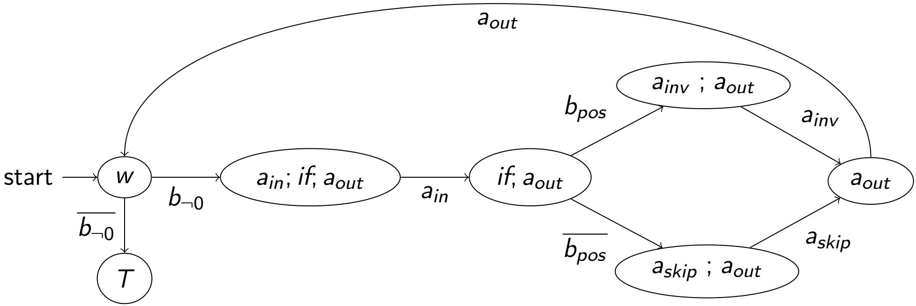
$$w := \quad \textbf{while } b_{\neg 0} \textbf{ do } \begin{array}{l} \text{IN } x; \\ \textbf{if } (b_{pos}) \textbf{ then } a_{inv} \textbf{ else } a_{skip}; \\ \text{OUT } x \end{array}$$

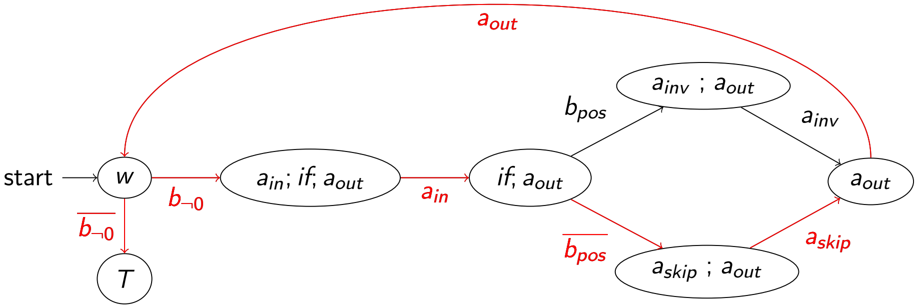
$$[x \mapsto 42], [\mathrm{w}] \rightarrow [x \mapsto 42], [\text{IN } x, \mathrm{w}] \xrightarrow{-1} \ldots \xrightarrow{1} \ldots \xrightarrow{5} \ldots \xrightarrow{5} \ldots$$

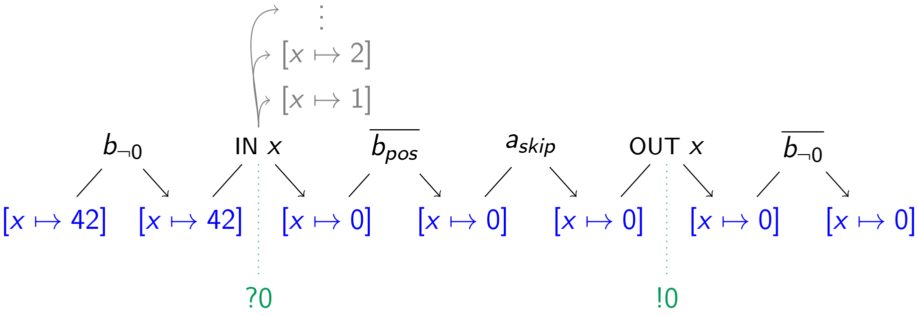
- Infinite traces should be observed
  - $\Rightarrow$ Partial traces / Prefixes
- Resulting traces: $\{\epsilon, \ldots, \text{-1}, \text{-1.1}, \text{-1.1.5}, \text{-1.1.5.5}, \ldots\}$
- End marker for terminating traces:
  - $\Rightarrow$ $\text{-1.1.5.5.0.0.\#} \in \mathrm{TS}_P(w, [x \mapsto 42])$

Abstraction from States

- Recording all actions instead of only effects allows to reconstruct
  - final states
  - effects
- Tests are modelled as partial identities on states




a_{out}
start
w
$b_{\neg 0}$
$\overline{b_{\neg 0}}$
T
a_{in}; if; a_{out}
a_{in}
if; a_{out}
b_{pos}
$\overline{b_{pos}}$
a_{inv} ; a_{out}
a_{skip} ; a_{out}
a_{inv}
a_{skip}
a_{out}


$[x \mapsto 2]$
$[x \mapsto 1]$
$b_{\neg 0}$
IN x
$\overline{b_{pos}}$
a_{skip}
OUT x
$\overline{b_{\neg 0}}$
$[x \mapsto 42]$
$[x \mapsto 42]$
$[x \mapsto 0]$
$[x \mapsto 0]$
$[x \mapsto 0]$
$[x \mapsto 0]$
$[x \mapsto 0]$
?0
!0

Regular Programs (RP)

Connection with regular expressions

$$s, t \ ::= \ \emptyset \ \mid \ \epsilon \ \mid \ a \ \mid \ s; t \ \mid \ s + t \ \mid \ s^*$$

Imp programs as regular programs:

$$\textbf{if } b \textbf{ then } s \textbf{ else } t \leadsto b; s + \overline{b}; t$$

$$\textbf{while } b \textbf{ do } s \leadsto (b; s)^*; \overline{b}$$

Example

$$(b_{\neg 0}; a_{in}; (b_{pos}; a_{inv} + \overline{b_{pos}}; a_{skip}); a_{out})^*; \overline{b_{\neg 0}}$$

Trace Semantics for RE (with stacks)

$$\frac{}{\epsilon / T} \qquad \frac{}{\# / [\,]} \qquad \frac{\xi / T}{a.\xi / a :: T} \qquad \frac{\xi / s :: t :: T}{\xi / (s;t) :: T} \qquad \frac{\xi / s :: T}{\xi / (s+t) :: T}$$

$$\frac{\xi / t :: T}{\xi / (s+t) :: T} \qquad \frac{\xi / T}{\xi / s^* :: T} \qquad \frac{\xi / s :: s^* :: T}{\xi / s^* :: T}$$

- Abort-rule for partial traces
- Marker-rule for terminal traces
- Two rules for $*$
  - Rule to finish
  - Rule to iterate

$$\xi \text{ executable on } \sigma := \ \xi = a_0.a_1.\ldots.a_n$$
$$\wedge \exists \sigma_0, \sigma_1, \ldots, \sigma_{n+1}, \forall i \leq n, \text{exec}(\sigma_i, a_i, \sigma_{i+1}) \wedge \sigma_0 = \sigma$$

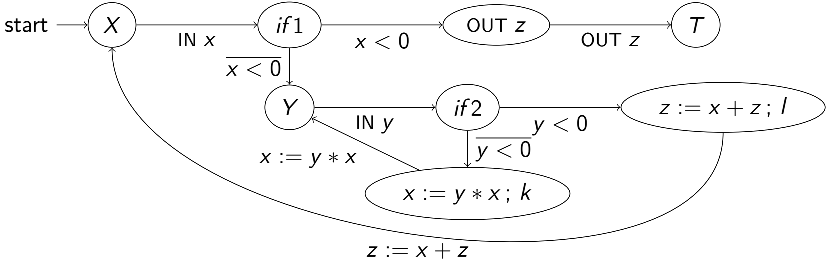
Linear Imp

$$s, t \ ::= \ a\,;\, s \ \mid \ \textbf{if } b \textbf{ then } s \textbf{ else } t \ \mid \ \textbf{fix } X.\, s \ \mid \ X$$

LImp no subset of regular programs

Example

$$\begin{aligned}
&\textbf{fix } X.\, \mathrm{IN}\ x;\ \textbf{if } x < 0 \textbf{ then } \mathrm{OUT}\ z \\
&\qquad\qquad \textbf{else fix } Y.\, \mathrm{IN}\ y;\ \textbf{if } y < 0 \textbf{ then } z := x + z\,;\ X \\
&\qquad\qquad\qquad\qquad \textbf{else } x := y * x\,;\ Y
\end{aligned}$$



Context-Free Programs (CFP)

$$s, t \;::=\; \epsilon \mid a \mid s; t \mid s + t \mid \mathrm{fix}\, X.\, s \mid X$$

$$\frac{\xi \big/ s^{X}_{\mathrm{fix}\, X.\, s} :: T}{\xi \big/ (\mathrm{fix}\, X.\, s) :: T}$$

Not only regular languages:

Example

$$s := \quad \mathrm{fix}\, x.\, \epsilon + a; x; b$$

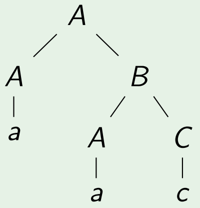
$$\forall n \in \mathbb{N}.\; a^n b^n \# \big/ s$$

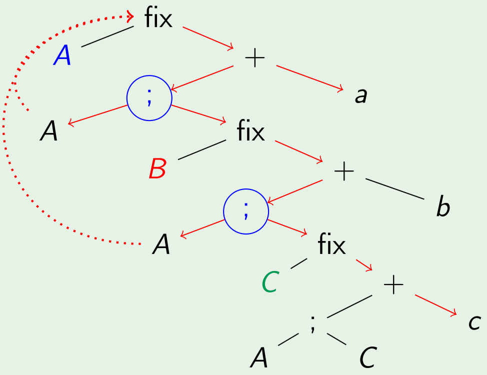
Can express Context-Free Grammars

Example

$$A \mapsto AB \mid a$$
$$B \mapsto AC \mid b$$
$$C \mapsto AC \mid c$$

Example: a a c





$$\text{fix}\, A.\ A; (\text{fix}\, B.\ A; (\text{fix}\, C.\ A; C + c) + b) + a$$

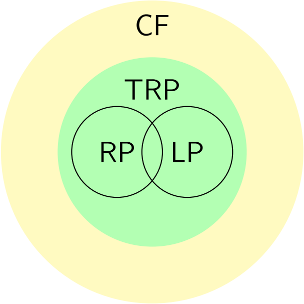
Tail-Recursive Programs (TRP)

tail-recursive s := All bound variables in s occur in end position

regular s := Every fix in s is of the form $\mathrm{fix}\,X.\,\epsilon + t; X$ or $\mathrm{fix}\,X.\,X$

linear s := Every sequence in s is of the form $a; s$

- Regular programs (RP) are tail-recursive (especially Imp)
- Linear programs (LP) are tail-recursive (especially LImp)

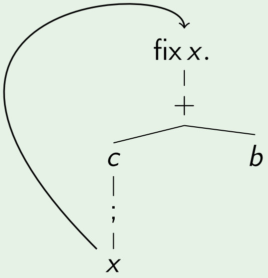


From Tail-Recursive Programs to Regular Programs

Tail-recursive programs can be translated in regular programs

Example

$$\mathsf{fix}\, x.\, c; x + b$$



$$\longrightarrow \quad c^* b$$

Intuition: Every fix can be split up in a part that will iterate and one which will finish

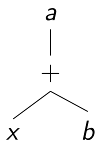
Regularizer - Example

$$\mathcal{R} : \mathsf{trp} \to (\mathcal{V} \times \mathsf{rp})\ \mathsf{list} \times \mathsf{rp}$$

Example

$$\mathsf{fix}\, x.\, a; (x + b)$$

$\mathcal{R}$



$$= \begin{bmatrix} \_ \ : \ a; b \\ x \ : \ a; \epsilon \end{bmatrix}$$

$$\mathcal{R}\ (\mathsf{fix}\, x.\, a; (x + b)) = \begin{bmatrix} \_ \ : \ (a; \epsilon)^*; a; b \end{bmatrix}$$

Normalizing expressions

- Goal: Equations which allow to transform TRP stepwise to RP
- Searched: Adequate equivalence $\approx$ which allows the following transformations:

For $\mathrm{fix}\, X.\, s$ split up the function body s in a disjunction of the form

$$s \approx s' + (t; x)$$

where x is not in the free variables of s' and t contains no free variables
With

$$t^* = \mathrm{fix}\, X.\, \epsilon + t; X$$

We show:

$$\mathrm{fix}\, X.\, s \approx t^*; s'$$

Program Equivalence

$$s \approx t := \forall \xi.\ {}^{\xi}\!/[s] \leftrightarrow {}^{\xi}\!/[t]$$

Problem: $\approx$ not congruent for fix

Example

$$a; x \approx a; y \qquad \text{but} \qquad \text{fix}\, x.\, a; x \not\approx \text{fix}\, x.\, a; y$$

Possible solutions:

- Redefining equivalence with environment $\alpha : \mathcal{V} \to \text{cfp}$:

$$s \equiv t := \ \forall \alpha\, \xi.\ \ {}^{\xi}\!/_{s,\alpha} \leftrightarrow {}^{\xi}\!/_{t,\alpha}$$

- Axiomatic characterization with a depth counter:

$$s \equiv_n t := \quad s \approx t \text{ and only calls to variables} > n \text{ may differ}$$

Linearizer

Linear context-free programs (LP) as a generalization of LImp:

$$\text{LImp} \qquad s, t \ ::= \ a \,;\, s \ \mid \ \textbf{if } b \textbf{ then } s \textbf{ else } t \ \mid \ \textbf{fix}\, X \,.\, s \ \mid \ X$$

$$\text{LP} \qquad s, t \ ::= \ \epsilon \mid a; s \mid s + t \mid \text{fix}\, X \,.\, s \mid X$$

Translation from TRP to LP:

$$\begin{aligned}
\mathcal{L}(_, _) &: \text{trp} \to \text{lp} \to \text{lp} \\
\mathcal{L}(\epsilon, u) &= \epsilon \\
\mathcal{L}(x, u) &= x \\
\mathcal{L}(a, u) &= a; u \\
\mathcal{L}(s; t, u) &= \mathcal{L}(s, \mathcal{L}(t, u)) \\
\mathcal{L}(s + t, u) &= \mathcal{L}(s, u) + \mathcal{L}(t, u) \\
\mathcal{L}(\text{fix}\, x \,.\, s, u) &= \text{fix}\, X \,.\, \mathcal{L}(s, u) \qquad \text{non-capturing}
\end{aligned}$$

Correctness Proof

Verification with respect to $\approx$

$$s \approx t := \forall \xi.\ {}^{\xi}\!/[s] \leftrightarrow {}^{\xi}\!/[t]$$

Strong substitution lemma needed which allows to substitute under fix:

$$s \text{ trp} \rightarrow \mathcal{L}(s^x_t, u) = \mathcal{L}(s, u)^x_{\mathcal{L}(t,u)}$$

Proof by induction on s trp

Correctness Lemma:

$$(\forall s \in T, \text{trp } s) \rightarrow {}^{\xi_1}\!/T \rightarrow {}^{\xi_2}\!/[u] \rightarrow {}^{\xi_3}\!/T' \rightarrow {}^{\xi_1 \cdot \xi_2 \cdot \xi_3}\!/(\mathcal{L}(T,u)) :: T'$$

$$\mathcal{L}(T, u) := \text{foldr } (\lambda(s, \text{cont}).\ \mathcal{L}(s, \text{cont}))\ u\ T$$

Program Equivalence

Verification of translation with program equivalence

$$s \equiv t := \forall \alpha\, \xi.\ \ {}^{\xi}\!/_{s,\alpha} \leftrightarrow {}^{\xi}\!/_{t,\alpha}$$

Characteristic equations to push sequences inwards:

$$\epsilon; u \equiv u$$

$$(s; t); u \equiv s; (t; u)$$

$$(s + t); u \equiv s; u + t; u$$

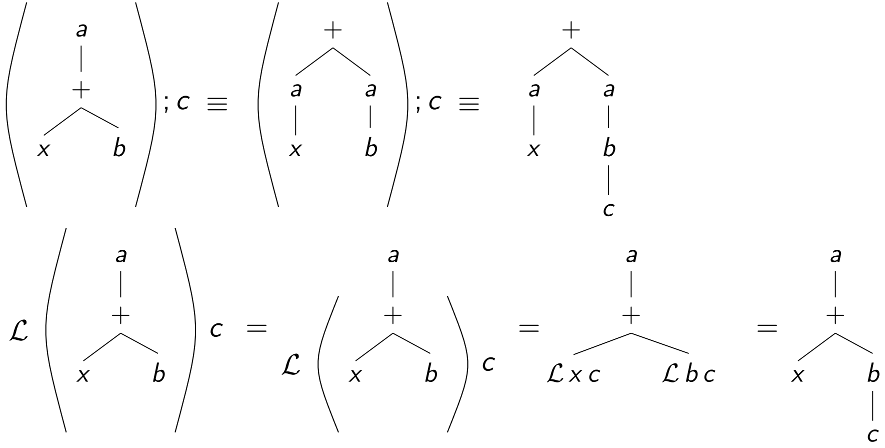
$$(\mathrm{fix}\, X.\, s + (t; X)); u \equiv \mathrm{fix}\, X.\, (s; u) + (t; X)$$

if X is not in the free variables of s u and t

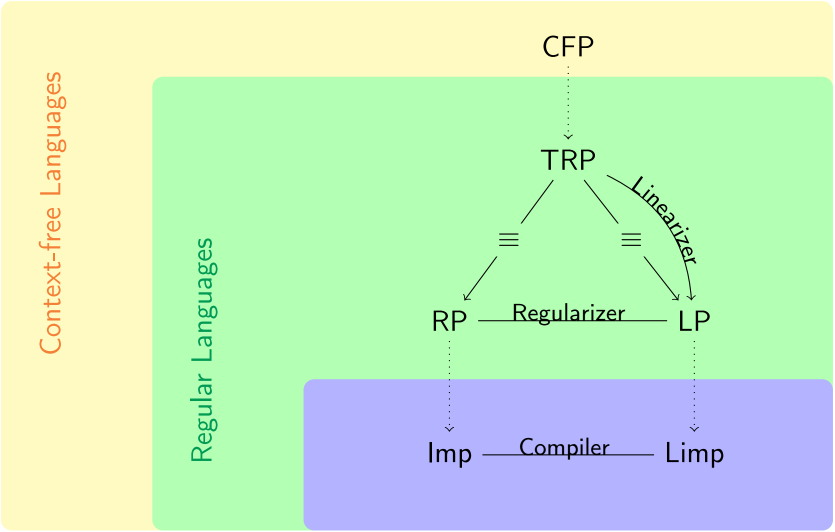
Linearizer - Example

Example

$$(\text{fix}\, x.\, a; (x + b)); c$$



Summary


Context-free Languages
Regular Languages
CFP
TRP
Linearizer
≡
≡
RP
Regularizer
LP
Imp
Compiler
Limp

Results

Results obtained:

- Linearizer for TRP
- Correctness of the TRP-Linearizer with trace semantics
- Regularizer

Results wanted:

- Correctness of Regularizer
- Correctness of characteristic fix-equations
- Correctness results with stronger semantics ($\equiv$)
- Sound equational system for $\equiv$

Outlook

- Correspondence of context-free programs and context-free languages (together with Jana)
- Connection between partial trace equivalence and big-step semantics
- Decidability of prefix languages for tail-recursive programs
- Equational deduction system for tail-recursive programs
- Extending context-free programs with mutual recursion

4 Appendix

Tail-Recursion

$$\frac{}{\mathsf{trc}\ x\ \epsilon} \qquad \frac{}{\mathsf{trc}\ x\ a} \qquad \frac{}{\mathsf{trc}\ x\ y} \qquad \frac{\mathsf{trc}\ x\ s \qquad \mathsf{trc}\ x\ t}{\mathsf{trc}\ x\ (s + t)}$$

$$\frac{x \notin \mathcal{F}(s) \qquad \mathsf{trc}\ x\ t}{\mathsf{trc}\ x\ (s; t)} \qquad \frac{\mathsf{trc}\ x\ s}{\mathsf{trc}\ x\ (\mathsf{fix}\, y.\, s)} \qquad \frac{x \in \mathcal{F}(s)}{\mathsf{trc}\ x\ s}$$