Regularity and Linearization of Tail-recursive programs

Final Bachelor Talk

Clara Schneidewind
Advisor: Prof. Dr. Gert Smolka

October 30th, 2015

Motivation

Example (IMP)

while $(x \geq n)$

$x := x - n$

end ;

OUT x

Example (Linear IMP)

block l :

if $(x \geq n)$

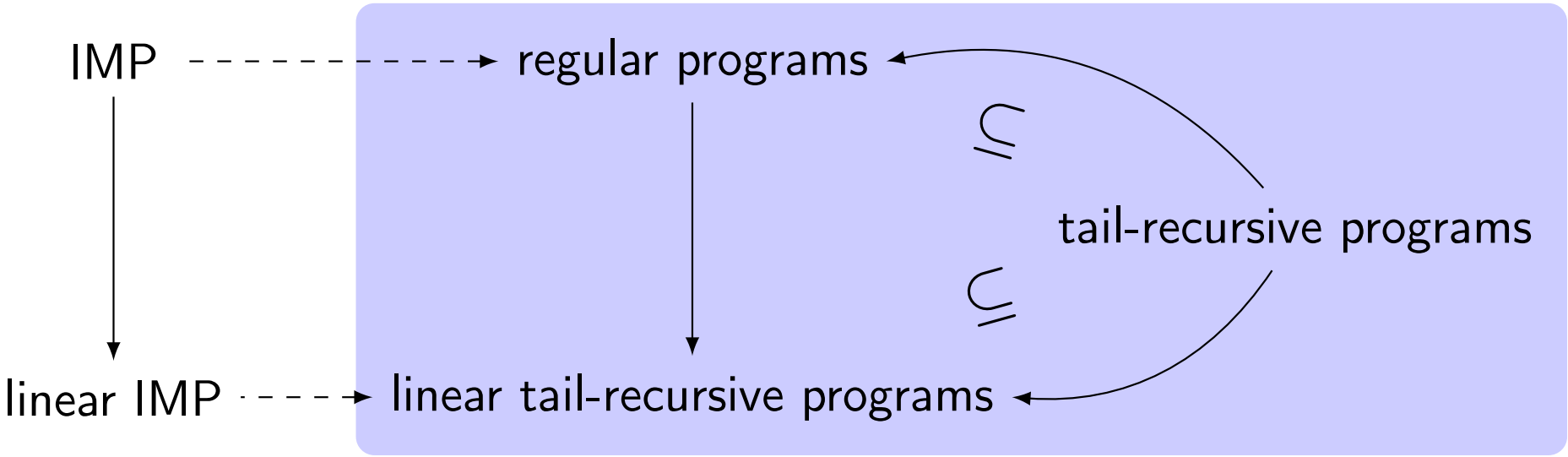
then $x := x - n$; **call** l

else OUT x

call l

- IMP: idealized imperative programming language (Winskel)
- Linear IMP: idealized imperative intermediate language
  - blocks instead of loops
  - no full sequential composition (form: $x := a;\ c$)

Overview



context-free programs

Content

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

Context-free programs

$$s, t, u \ ::= \ 1 \ \mid \ x \ \mid \ s\,t \ \mid \ s + t \ \mid \ \mu x.s \qquad (x \in \mathbb{N})$$

- no variable assignments and conditions
- free variables as abstract actions
- non-determinism
- recursion operator

Example

1	skip
x	program invoking action x
$\mu x.x \quad (\varnothing)$	silently diverging program
$\mu x.1 + a\,x \quad (a^*)$	program iterating a

Traces

Trace := sequence of variables with ending

$$\xi, \eta, \zeta := \epsilon \mid \# \mid x\xi \qquad (x \in \mathbb{N})$$

Trace concatenation

$$\epsilon\,\eta := \epsilon$$
$$\#\,\eta := \eta$$
$$(x\xi)\,\eta := x(\xi\,\eta)$$

Note: partial traces are absorbing:

Example

$$(xy\epsilon)(z\#) = xy\epsilon$$

Trace semantics

$\xi/s := \xi$ is a trace of the program s

$$\frac{}{\epsilon/s} \qquad \frac{}{\#/1} \qquad \frac{\xi = \epsilon \vee \xi = \#}{x\xi/x} \qquad \frac{\xi_1/s \qquad \xi_2/t}{\xi_1\,\xi_2/s\,t} \qquad \frac{\xi/s}{\xi/s+t}$$

$$\frac{\xi/t}{\xi/s+t} \qquad \frac{\xi/s^x_{\mu x.s}}{\xi/\mu x.s}$$

$$\mathcal{T}s := \{\xi \mid \xi/s\}$$

Example

$$\mathcal{T}1 = \{\epsilon, \#\} \quad \mathcal{T}(x\,y) = \{\epsilon, x\epsilon, xy\epsilon, xy\#\} \quad \mathcal{T}(\mu x.x) = \{\epsilon\}$$
$$\mathcal{T}(\mu x.1 + a\,x) = \{\epsilon, a\epsilon, a\#, aa\epsilon, aa\#, \dots\}$$

Program equivalence

Program equivalence

$$s \approx t := \mathcal{T}\, s = \mathcal{T}\, t$$

Important equivalences

$$(s + t) + u \approx s + (t + u) \qquad s + t \approx t + s$$
$$\varnothing + s \approx s \qquad s + s \approx s$$
$$s\,(t\,u) \approx (s\,t)\,u \qquad 1\,s \approx s$$
$$s\,1 \approx s \qquad s\,(t + u) \approx s\,t + s\,u$$
$$(s + t)\,u \approx s\,u + t\,u \qquad \varnothing\,s \approx \varnothing$$

Compatitibility with program structure

$$\frac{s \approx s' \qquad t \approx t'}{s\,t \approx s'\,t'} \qquad \frac{s \approx s' \qquad t \approx t'}{s + t \approx s' + t'} \qquad \frac{s \approx s'}{\mu x.s \approx \mu x.s'}$$

Program classes

- Tail-recursive programs
  - Bound variables only in tail position

Example

$$1 \qquad x \qquad \mu x.x \qquad (\mu x.x)\,1 \qquad \mu x.1 + a\,x \qquad \mu x.x + (\mu y.x + y)$$

- Regular programs
  - recursions restricted to the following forms:

$$\varnothing := \mu x.x \qquad \textit{(null program)}$$

$$s^* := \mu x.1 + s\,x \qquad x \text{ not free in } s \qquad \textit{(iteration)}$$

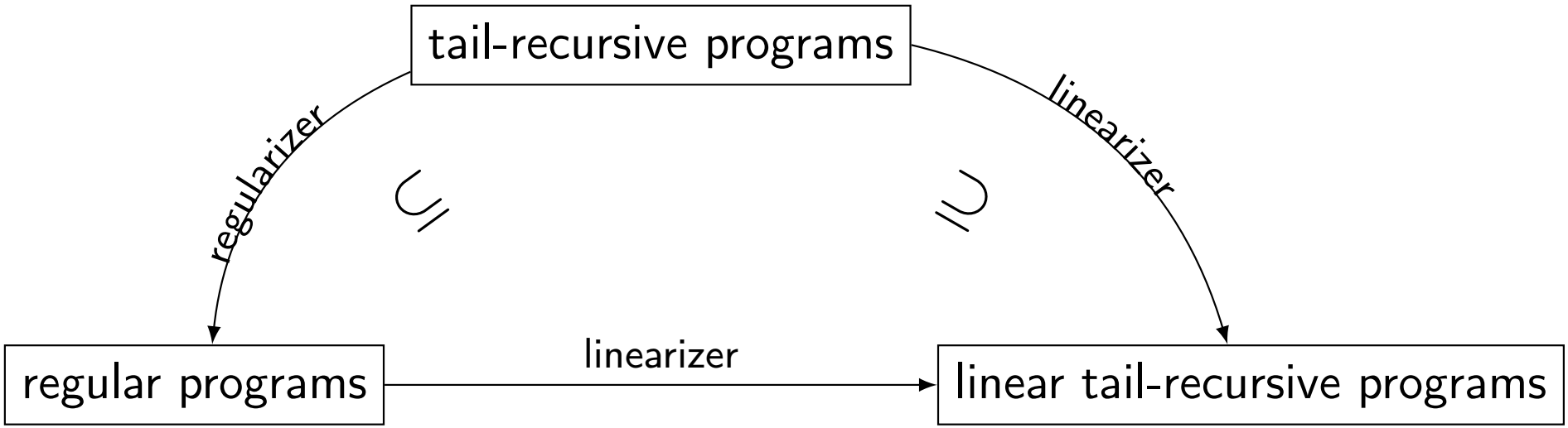
- Linear tail-recursive programs programs
  - Sequential composition restricted to the form $x\,s$.

Example (Non-linear tail-recursive programs)

$$(x\,y)\,z \qquad (x + y)\,z \qquad (\mu x.x)\,1$$

Transformations

Regular programs and linear tail-recursive programs are tail-recursive



Regularization

- Regularization: Transforming recursions into iterations
- Regular bodies of recursions can be decomposed into an iterating and a terminating part

Example

$$\mu x.1 \approx \mu x.1 + \varnothing\, x$$
$$\mu x.a\, x \approx \mu x.\varnothing + a\, x$$
$$\mu x.a^*\, (x + b) \approx \mu x.a^*\, b + a^*\, x$$

- Decomposed recursions can be transformed to iterations

$$\mu x.s + t\, x \approx t^*\, s \qquad x \text{ not free in } s \text{ and } t$$

Regularization - Example

Recursions are dissolved in a bottom-up fashion

Example

$$
\begin{aligned}
&\mu x.a + (\mu y.\underbrace{b\,x + c\,y + d}_{\approx (b\,x+d)+c\,y}) \\
&\approx \mu x.a + (\underbrace{\mu y.(b\,x + d) + c\,y}_{\approx c^*\,(b\,x+d)}) \\
&\approx \mu x.\underbrace{a + c^*\,(b\,x + d)}_{\approx (a+c^*\,d)+c^*\,b\,x} \\
&\approx \underbrace{\mu x.(a + c^*\,d) + c^*\,b\,x}_{\approx (c^*\,b)^*\,(a+c^*\,d)} \\
&\approx (c^*\,b)^*\,(a + c^*\,d)
\end{aligned}
$$

Regularizer

- recursively transforms decomposed recursions to iterations
- uses a decomposer $D : \mathsf{var} \to \mathsf{cfp} \to \mathsf{cfp} \times \mathsf{cfp}$ to decompose regular bodies into their iterating and terminating parts

Regularizer

$$
\begin{aligned}
R &: \mathsf{cfp} \to \mathsf{cfp} \\
R\,1 &:= 1 \\
R\,x &:= x \\
R\,(s\,t) &:= (R\,s)\,(R\,t) \\
R\,(s+t) &:= R\,s + R\,t \\
R\,(\mu x.s) &:= \mathsf{let}\,(s_1, s_2) := D_x\,(R\,s)\,\mathsf{in}\,s_2^*\,s_1
\end{aligned}
$$

Linearization of regular programs

Linearization: Dissolving full sequential composition

$$1\,u \approx u$$
$$(s\,t)\,u \approx s\,(t\,u)$$
$$(s+t)\,u \approx s\,u + t\,u$$
$$\varnothing\,u \approx \varnothing$$
$$s^*\,u = (\mu x.1 + s\,x)\,u \approx \mu x.u + s\,x \qquad x \text{ not free in } s \text{ and } u$$

Example

$$\begin{aligned}(a+b)^*\,c &= (\mu x.1 + (a+b)\,x)\,c \\ &\approx \mu x.c + (a+b)\,x \\ &\approx \mu x.c + a\,x + b\,x\end{aligned}$$

Linearizer for tail-recursive programs

- Goal: structure preserving linearization
- Idea: remembering bound variables + appending continuation to non-iterating parts

$$
\begin{aligned}
L &: \mathrm{var}\ \mathit{set} \to \mathrm{cfp} \to \mathrm{cfp} \to \mathrm{cfp} \\
L_V\, 1\, u &:= u \\
L_V\, x\, u &:= \mathit{if}\ x \in V\ \mathit{then}\ x\ \mathit{else}\ x\ u \\
L_V\, (s\ t)\, u &:= L_V\, s\, (L_V\, t\, u) \\
L_V\, (s + t)\, u &:= L_V\, s\, u + L_V\, t\, u \\
L_V\, (\mu x.s)\, u &:= \mu x.L_{V \cup \{x\}}\, s\, u \qquad\qquad x \text{ not free in } u
\end{aligned}
$$

Linearizer - Example

Example

$$\begin{aligned} L_\emptyset\,(\mu x.a\,(b+x))\,c &= \mu x.L_{\{x\}}\,(a\,(b+x))\,c \\ &= \mu x.L_{\{x\}}\,a\,(L_{\{x\}}\,(b+x)\,c) \\ &= \mu x.L_{\{x\}}\,a\,(L_{\{x\}}\,b\,c + L_{\{x\}}\,x\,c) \\ &= \mu x.L_{\{x\}}\,a\,(b\,c+x) \\ &= \mu x.a\,(b\,c+x) \end{aligned}$$

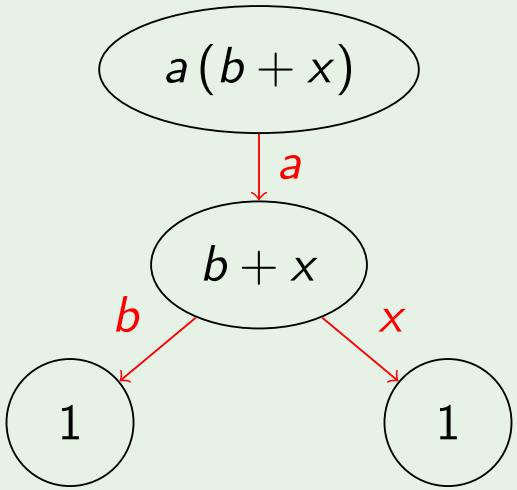
Correctness statement for the linearizer:

$$\left.\begin{array}{l}\text{tail-recursive } s \\ \text{linear tail-recursive } u\end{array}\right\} L_\emptyset\,s\,u \approx s\,u \wedge \text{linear tail-recursive } L_\emptyset\,s\,u$$

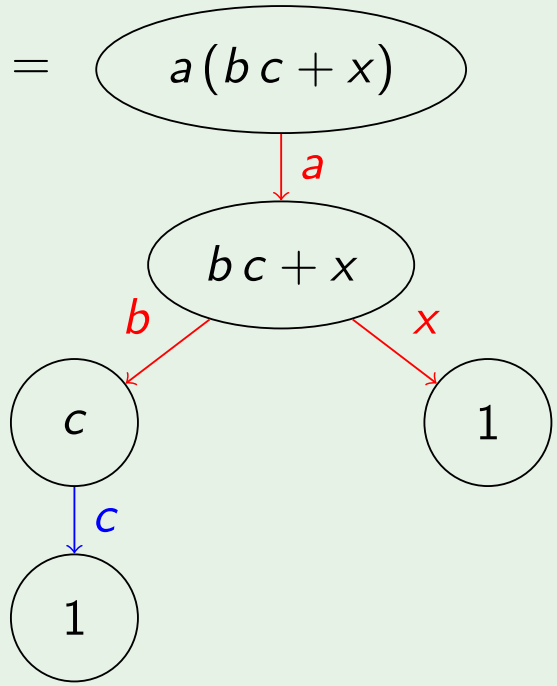
Linearizer - Correctness

$$\xi/L_V\, s\, u \begin{cases} 1. & \xi \text{ is a partial trace of } s \text{ and } x \in V \text{ do not occur in } \xi \\ 2. & \xi \text{ is a trace of } s \text{ and ends with } x \in V \\ 3. & \xi = \xi_1\xi_2,\ \xi_1 \text{ is a total trace of } s \text{ and } x \in V \\ & \text{do not occur in } \xi_1 \text{ and } \xi_2 \text{ is a trace of } u \end{cases}$$

Example



$L_{\{x\}}\,(a\,(b+x))\,c =$



$\xi \in \{a\epsilon, ab\epsilon, abb\epsilon, abx\#, ax\#\}$

$\xi_2 \in \{\epsilon, c\epsilon, c\#\}$

Abstract IMP

$$c, d \ ::= \ \text{skip} \ \mid \ a \ \mid \ c; d \ \mid \ \text{if } b \ c \ d \ \mid \ \text{while } b \ c \quad (a \in \mathcal{A}) \quad (b \in \mathcal{B})$$

- $\mathcal{A}$: set of of abstract actions
- actions change the state
- $\mathcal{B} \subseteq \mathcal{A}$: set of tests closed under negation ($\overline{b}$)
- tests are partial identities on the state

Context-free abstractions (cfa)

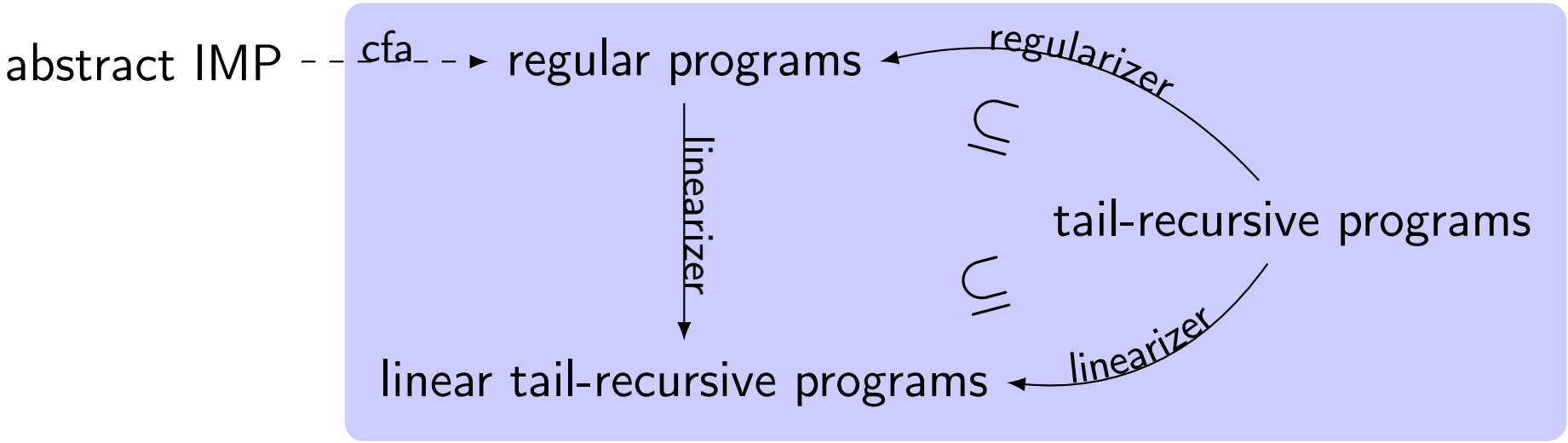
$$\text{if } b \ c \ d \rightsquigarrow b\,c + \overline{b}\,d$$

$$\text{while } b \ c \rightsquigarrow \mu x.\overline{b} + b\,c\,x$$

Correspondence

$$\text{cfa } c \approx \text{cfa } d \rightarrow c \approx_{\text{IMP}} d$$

Summary



context-free programs

- Abstraction from imperative programs
- Verification of program transformations with respect to trace semantics

Future work

- Context-free abstractions for reactive IMP
- Total traces of context-free programs describe context-free languages (extend context-free programs with mutual recursion)
- Brzozowski derivatives for tail-recursive context-free programs (deciding program equivalence)

7 Appendix

Tail recursion

Tail recursion

$$\frac{}{\mathrm{trec}_V\, 1} \qquad \frac{}{\mathrm{trec}_V\, x} \qquad \frac{V \parallel \mathcal{V}s \qquad \mathrm{trec}_\emptyset\, s \qquad \mathrm{trec}_V\, t}{\mathrm{trec}_V\,(s\, t)}$$

$$\frac{\mathrm{trec}_V\, s \qquad \mathrm{trec}_V\, t}{\mathrm{trec}_V\,(s + t)} \qquad \frac{\mathrm{trec}_{(V \cup \{x\})}\, s}{\mathrm{trec}_V\,(\mu x.s)}$$

Linear tail recursion

$$\frac{}{\mathrm{ltrec}_V\, 1} \qquad \frac{}{\mathrm{ltrec}_V\, x} \qquad \frac{x \notin V \qquad \mathrm{ltrec}_V\, s}{\mathrm{ltrec}_V\,(x\, s)} \qquad \frac{\mathrm{ltrec}_V\, s \qquad \mathrm{ltrec}_V\, t}{\mathrm{ltrec}_V\,(s + t)}$$

$$\frac{\mathrm{ltrec}_{(V \cup \{x\})}\, s}{\mathrm{ltrec}_V\,(\mu x.s)}$$

Regularity

Regularity - version 1

$$\frac{}{\mathrm{reg}\,1} \qquad \frac{}{\mathrm{reg}\,x} \qquad \frac{\mathrm{reg}\,s \qquad \mathrm{reg}\,t}{\mathrm{reg}\,(s\,t)} \qquad \frac{\mathrm{reg}\,s \qquad \mathrm{reg}\,t}{\mathrm{reg}\,(s+t)} \qquad \frac{x \notin \mathcal{V}s \qquad \mathrm{reg}\,s}{\mathrm{reg}\,(\mu x.1+s\,x)}$$

$$\frac{}{\mathrm{reg}\,\varnothing}$$

Regularity - Version 2

$$\frac{}{\mathrm{reg}_V\,1} \qquad \frac{x \notin V}{\mathrm{reg}_V\,x} \qquad \frac{\mathrm{reg}_V\,s \qquad \mathrm{reg}_V\,t}{\mathrm{reg}_V\,(s\,t)} \qquad \frac{\mathrm{reg}_V\,s \qquad \mathrm{reg}_V\,t}{\mathrm{reg}_V\,(s+t)} \qquad \frac{}{\mathrm{reg}_V\,\varnothing}$$

$$\frac{\mathrm{reg}_{(V\cup\{x\})}\,s}{\mathrm{reg}_V\,(\mu x.1+s\,x)}$$

Abstract IMP

$$c, d \ ::= \ \text{skip} \mid a \mid c; d \mid \text{if } b\ c\ d \mid \text{while } b\ c \quad (a \in \mathcal{A}) \quad (b \in \mathcal{B})$$

- Σ: abstract states
- $\mathcal{A}$: set of of abstract actions
- $\mathcal{B} \subseteq \mathcal{A}$: set of tests closed under negation ($\overline{b}$)
- Execution predicate $\dot{\rightarrow}$: $\Sigma \times \mathcal{A} \times \Sigma$
- $\sigma \xrightarrow{b} \tau \rightarrow \sigma = \tau$
- $\sigma \xrightarrow{\overline{b}} \sigma \leftrightarrow \neg\sigma \xrightarrow{b} \sigma$

} tests as partial identities on the state

Big-step semantics for abstract IMP

$$\frac{}{(\sigma, \mathsf{skip}) \Rightarrow \sigma} \qquad \frac{\sigma \xrightarrow{a} \tau}{(\sigma, a) \Rightarrow \tau} \qquad \frac{\sigma \xrightarrow{b} \sigma \qquad (\sigma, c) \Rightarrow \tau}{(\sigma, \mathsf{if}\ b\ c\ d) \Rightarrow \tau}$$

$$\frac{\neg\sigma \xrightarrow{b} \sigma \qquad (\sigma, d) \Rightarrow \tau}{(\sigma, \mathsf{if}\ b\ c\ d) \Rightarrow \tau}$$

$$\frac{\sigma \xrightarrow{b} \sigma \qquad (\sigma, c) \Rightarrow \sigma' \qquad (\sigma', \mathsf{while}\ b\ c) \Rightarrow \tau}{(\sigma, \mathsf{while}\ b\ c) \Rightarrow \tau} \qquad \frac{\neg\sigma \xrightarrow{b} \sigma}{(\sigma, \mathsf{while}\ b\ c) \Rightarrow \sigma}$$

$$c \approx_{\mathsf{IMP}} d := \forall \sigma \tau.\ (\sigma, c) \Rightarrow \tau \leftrightarrow (\sigma, d) \Rightarrow \tau$$

Encoding abstract IMP with context-free programs

Context-free abstractions (cfa)

$$\text{if } b\ c\ d \rightsquigarrow b\,s + \overline{b}\,t$$

$$\text{while } b\ c \rightsquigarrow \mu x.\overline{b} + b\,s\,x$$

Correspondence

$$\text{cfa } c \approx \text{cfa } d \to c \approx_{\text{IMP}} d$$

$$\sigma \xrightarrow{\xi} \tau :\leftrightarrow \xi = a_1, \ldots a_n\# \wedge \sigma \xrightarrow{a_1} \sigma_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} \tau$$

for some $\sigma_1, \ldots, \sigma_{n-1}$

$$(\sigma, c) \Rightarrow \tau \to \exists \xi.\ \xi/\text{cfa } c \wedge \sigma \xrightarrow{\xi} \tau$$

$$\xi/\text{cfa } c \to \sigma \xrightarrow{\xi} \tau \to (\sigma, c) \Rightarrow \tau$$