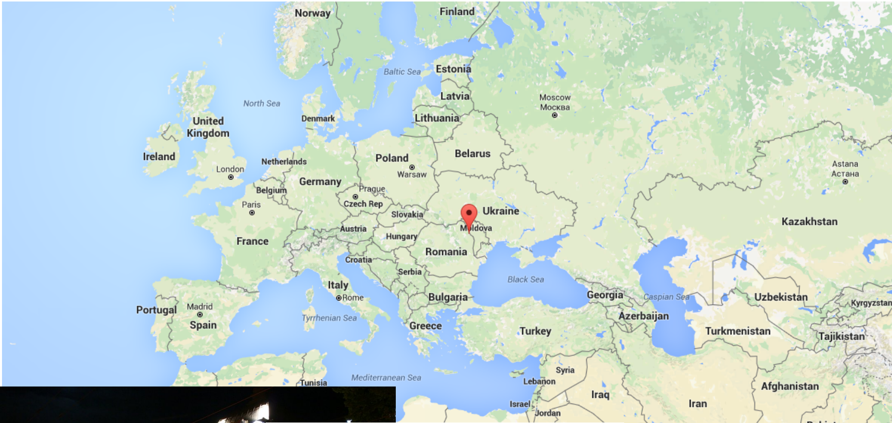
How to Prove Program Equivalence with Matching Logic

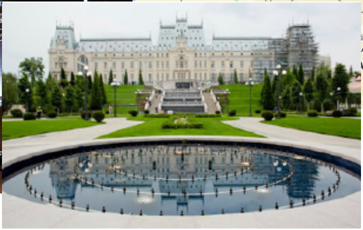
Ștefan Ciobâcă
"Alexandru Ioan Cuza" University, Iași, Romania

This presentation includes joint work with

Dorel Lucanu (UAIC, Iași, Romania), Vlad Rusu (INRIA Lille, France) and Grigore Roșu (UIUC, USA).

April 11th, 2015
Workshop on Program Equivalence
London, UK


Norway
Finland
Baltic Sea
Estonia
Latvia
Moscow
Москва
North Sea
United Kingdom
Denmark
Lithuania
Ireland
Netherlands
Poland
Belarus
Astana
Астана
London
Warsaw
Germany
Belgium
Prague
Paris
Czech Rep
Ukraine
Slovakia
Kazakhstan
Moldova
Austria
France
Hungary
Romania
Croatia
Serbia
Black Sea
Italy
Bulgaria
Georgia
Caspian Sea
Rome
Uzbekistan
Kyrgyzstan
Portugal
Madrid
Spain
Azerbaijan
Tyrrhenian Sea
Greece
Turkey
Turkmenistan
Tajikistan
Mediterranean Sea
Tunisia
Syria
Lebanon
Afghanistan
Iraq
Israel
Iran
Jordan
Pakistan
Persian Gulf
United Arab Emirates
Oman
Mumbai



The Plan

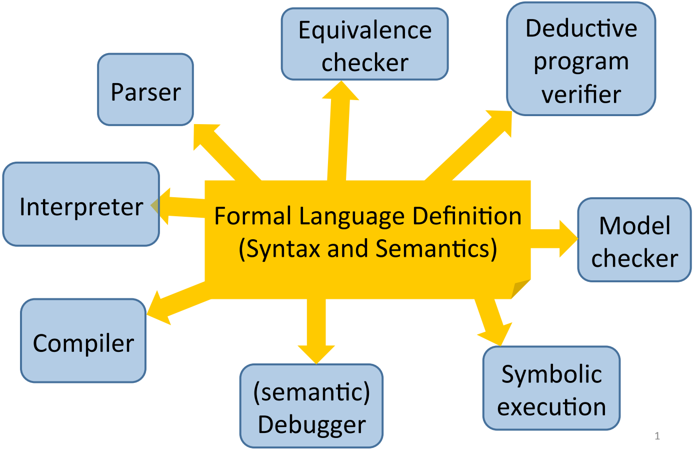
1. **Background**
2. Matching Logic
3. Reachability Logic
4. Language Aggregation
5. Proving Program Equivalence

Background - State-of-the-art in Program Verification

```
int main(void)
{
  int x = 0;
  return (x = 1) + (x = 2);
}
```

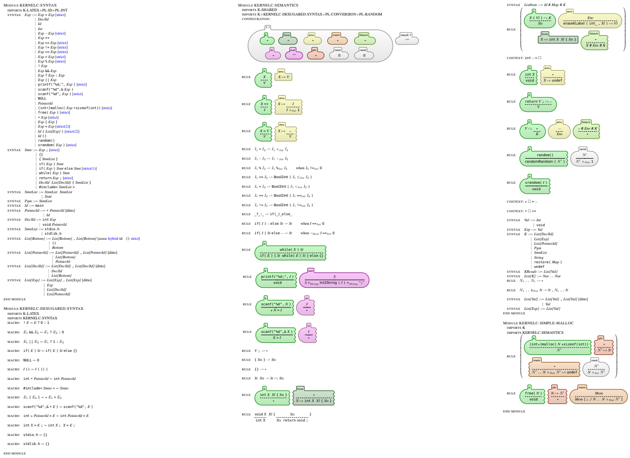
Jessie/Frama-C and Havoc prove the program returns. 4
GCC4, MSVC: the program returns. 4
GCC3, ICC, Clang: the program returns. 3
Reality: the program is undefined.

Formal Language Definition
(Syntax and Semantics)



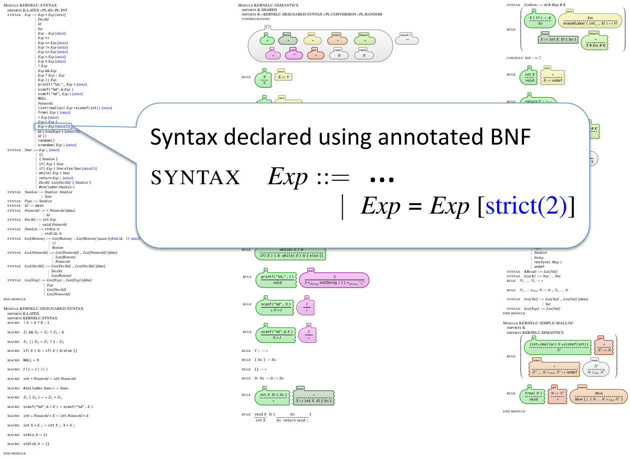
2005: Jose Meseguer and Grigore Roșu
The Rewriting Logic Semantics Project

Complete K Definition of KernelC



[Roșu, 2016]

Complete K Definition of KernelC

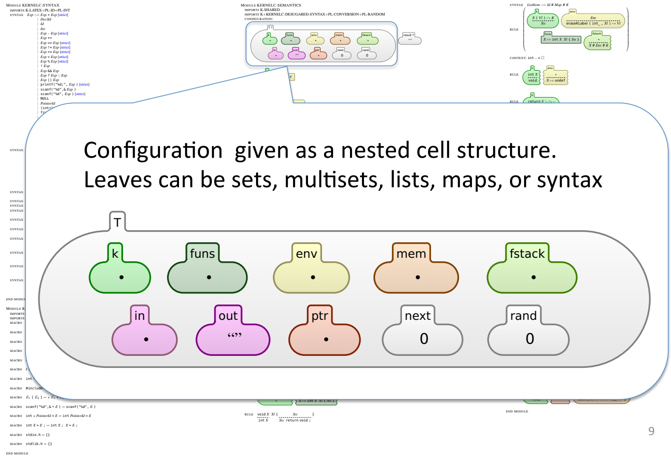


Syntax declared using annotated BNF

$$\textsc{syntax} \quad Exp ::= \ \ldots \qquad \mid \ Exp = Exp \ [\text{strict}(2)]$$

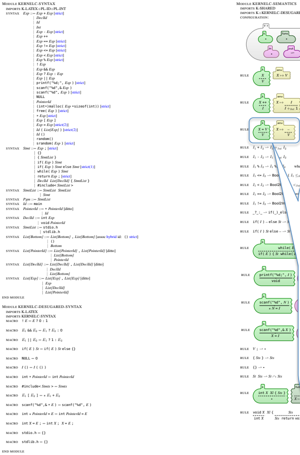
[Roșu, 2016]

Complete K Definition of KernelC

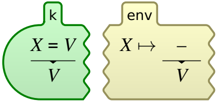


[Roșu, 2016]

Complete K Definition of KernelC



Semantic rules given contextually




```
rule
  <k> X = V => V ...</k>
  <env>... X |-> (_ => V) ...</env>
```


[Roșu, 2016]

Real Languages Defined in K

- Ellison et. al ([POPL 2012, PLDI 2015]) - C
- Bogdănaș et. al ([POPL 2015]) - Java 1.4
- Park et. al ([PLDI 2015]) - JavaScript

K Team

UIUC, USA

- Grigore Rosu (started K in 2003)
- Brandon Moore
- Daejun Park
- Lucas Pena
- Cosmin Radoi
- Yuwen Shijiao
- Andrei Stefanescu
  Former members: Kyle Blocher, Thomas Bogue, Peter Dinges, Chucky Ellison, Cansu Erdogan, Dwight Guth, Mike Ilseman, David Lazar, Patrick Meredith, Erick Mikida, Traian Serbanuta

Runtime Verification, Inc., USA

- Dwight Guth
- Manasvi Saxena

UAIC/UNIBUC, Romania

- Dorel Lucanu
- Traian Florin Serbanuta
- Andrei Arusoae
- Stefan Ciobaca
- Radu Mereuta
  Former Members: Irina Asavoae, Mihai Asavoae, Denis Bogdanas, Gheorghe Grigoras

The Plan

1. Background
2. **Matching Logic**
3. Reachability Logic
4. Language Aggregation
5. Proving Program Equivalence

Reasoning About Program Configurations - Matching Logic

- used to specify static properties of program configurations

$$\exists l.(\langle\langle \texttt{skip}\rangle_{code}, \langle \texttt{x} \mapsto l, \ldots\rangle_{env}, \langle l \mapsto i, \ldots\rangle_{heap}\rangle) \wedge i > 0$$

$$\langle\langle \texttt{if v then } s_1 \texttt{ else } s_2\rangle_{code}, \langle \texttt{v} \mapsto i\rangle_{env}, \langle \ldots\rangle_{heap}\rangle \wedge i \neq 0$$

Matching Logic = FOL + terms as atomic formulae

Reasoning about Configurations - Matching Logic

$$\varphi ::= \exists x.\varphi, \varphi \wedge \varphi, \neg\varphi, P(t_1, \ldots, t_n), \pi$$

$$\pi \in T_{Cfg}(Var)$$

- $(\gamma, \rho) \models \pi$, where π is a basic pattern if $\rho(\pi) = \gamma$.
- $(\gamma, \rho) \models \varphi_1 \wedge \varphi_2$ if $(\gamma, \rho) \models \varphi_1$ and $(\gamma, \rho) \models \varphi_2$
- $(\gamma, \rho) \models \neg\varphi'$ if it is not true that $(\gamma, \rho) \models \varphi'$
- $(\gamma, \rho) \models \exists x.\varphi'$, where x is of sort s, if there exists $e \in [\![s]\!]_{\mathcal{T}}$ such that $(\gamma, \rho[e/x]) \models \varphi'$ (where $\rho[e/x]$ is the valuation obtained from ρ by updating the value of x to be e).

$$\varphi = \langle \texttt{skip,}\ \ \texttt{s} \mapsto s, \texttt{n} \mapsto n \rangle \wedge s = n * (n+1)/2$$

$$(\langle \texttt{skip,}\ \ \texttt{s} \mapsto 6, \texttt{n} \mapsto 3 \rangle, \rho = (s/6, n/3)) \models \varphi$$

$$(\langle \texttt{skip,}\ \ \texttt{s} \mapsto 6, \texttt{n} \mapsto 3 \rangle, \rho = (s/10, n/4)) \not\models \varphi$$

$$(\langle \texttt{skip,}\ \ \texttt{s} \mapsto 6, \texttt{n} \mapsto 0 \rangle, \rho = (s/6, n/0)) \not\models \varphi$$

The Plan

1. Background
2. Matching Logic
3. **Reachability Logic**
4. Language Aggregation
5. Proving Program Equivalence

Reachability Logic

RL formula: $\varphi \Rightarrow \varphi'$

Semantics: any *terminating* $\gamma \in [\![\mathit{Cfg}]\!]_{\mathcal{T}}$ *that matches* φ *moves into* some $\gamma' \in [\![\mathit{Cfg}]\!]_{\mathcal{T}}$ matching φ'

- $\langle \texttt{skip;}s\texttt{,}\ \ \mathit{env} \rangle \Rightarrow \langle s\texttt{,}\ \ \mathit{env} \rangle$
- $\langle \texttt{if}\ \mathrm{i}\ \texttt{then}\ s_1\ \texttt{else}\ s_2\texttt{,}\ \ \mathit{env} \rangle \land i \neq 0 \Rightarrow \langle s_1\texttt{,}\ \ \mathit{env} \rangle$
- $\langle \texttt{SUM,}\ \ \texttt{s} \mapsto 0, \texttt{n} \mapsto n \rangle \Rightarrow \langle \texttt{skip,}\ \ \texttt{s} \mapsto n * (n+1)/2, \texttt{n} \mapsto n \rangle$

```
SUM = (s := 0
       i := 1
       while i <= n do
         s := s + i
         i := i + 1)
```

$$\frac{\begin{array}{c}\varphi \Rightarrow \varphi' \ \text{ if } \ \varphi_1 \Rightarrow \varphi'_1 \bigwedge \cdots \bigwedge \varphi_n \Rightarrow \varphi'_n \ \in \ \mathcal{A} \\ \psi \text{ is a structureless pattern} \\ \mathcal{A} \cup C \ \vdash \varphi_1 \wedge \psi \Rightarrow \varphi'_1 \quad \cdots \quad \mathcal{A} \cup C \ \vdash \varphi_n \wedge \psi \Rightarrow \varphi'_n\end{array}}{\mathcal{A} \ \vdash_C \varphi \wedge \psi \Rightarrow \varphi' \wedge \psi} \quad \textbf{Axiom :}$$

Reflexivity : $\mathcal{A} \ \vdash \varphi \Rightarrow \varphi$

Transitivity : $$\frac{\mathcal{A} \ \vdash_C \varphi_1 \Rightarrow \varphi_2 \qquad \mathcal{A} \cup C \ \vdash \varphi_2 \Rightarrow \varphi_3}{\mathcal{A} \ \vdash_C \varphi_1 \Rightarrow \varphi_3}$$

Consequence : $$\frac{\models \varphi_1 \rightarrow \varphi'_1 \qquad \mathcal{A} \ \vdash_C \varphi'_1 \Rightarrow \varphi'_2 \qquad \models \varphi'_2 \rightarrow \varphi_2}{\mathcal{A} \ \vdash_C \varphi_1 \Rightarrow \varphi_2}$$

Case Analysis : $$\frac{\mathcal{A} \ \vdash_C \varphi_1 \Rightarrow \varphi \qquad \mathcal{A} \ \vdash_C \varphi_2 \Rightarrow \varphi}{\mathcal{A} \ \vdash_C \varphi_1 \vee \varphi_2 \Rightarrow \varphi}$$

Abstraction : $$\frac{\mathcal{A} \ \vdash_C \varphi \Rightarrow \varphi' \qquad \text{where } X \cap FV(\varphi') = \emptyset}{\mathcal{A} \ \vdash_C \exists X\, \varphi \Rightarrow \varphi'}$$

Circularity : $$\frac{\mathcal{A} \ \vdash_{C \cup \{\varphi \Rightarrow \varphi'\}} \varphi \Rightarrow \varphi'}{\mathcal{A} \ \vdash_C \varphi \Rightarrow \varphi'}$$

(LICS 2013)

Proving Equivalence of Programs

Can we construct a program equivalence prover from the formal semantics of languages?

```
s := 0
i := 1
while i <= n do
  s := s + i
  i := i + 1
return s
```

P

```
let f n a =
  if n = 0 then a
  else f (n - 1) (a + n)
in
  f n 0
```

Q

Idea: reason directly about pair-programs $\langle P, Q \rangle$.

The Plan

1. Background
2. Matching Logic
3. Reachability Logic
4. **Language Aggregation**
5. Proving Program Equivalence

The Aggregation of Two PL Signatures

$$
\begin{tikzcd}
 & \boxed{(S_0, \Sigma_0)} \arrow[dl, "h_1"] \arrow[dr, "h_2"'] & \\
\boxed{(S_1, \Sigma_1)} \arrow[dr, "h'_1"] & & \boxed{(S_2, \Sigma_2)} \arrow[dl, "h'_2"'] \\
 & \boxed{(S', \Sigma')} \arrow[d, "\text{add } \langle \_, \_ \rangle"] & \\
 & \boxed{(S, \Sigma)} &
\end{tikzcd}
$$

Constructing the Aggregated Model

$$
\begin{tikzcd}
 & \boxed{M_0} \arrow[r, dash, dashed] & \boxed{(S_0, \Sigma_0)} \arrow[dl, "h_1"] \arrow[dr, "h_2"] & \\
\boxed{M_1} \arrow[ur, "h_1"] \arrow[r, dash, dashed] & \boxed{(S_1, \Sigma_1)} \arrow[dr, "h'_1"] & \boxed{M_2} \arrow[ul, "h_2"'] \arrow[r, dash, dashed] & \boxed{(S_2, \Sigma_2)} \arrow[dl, "h'_2"'] \\
 & \boxed{M'} \arrow[ul, "h'_1"'] \arrow[ur, "h'_2"] \arrow[r, dash, dashed] & \boxed{(S', \Sigma')} \arrow[d, "\text{add new sort } s \text{ and } \langle \_, \_ \rangle"] & \\
 & \boxed{M} \arrow[u] \arrow[r, dash, dashed] & \boxed{(S, \Sigma)} &
\end{tikzcd}
$$

Constructing the Aggregated Semantics

- $$\mathcal{A}_1 \otimes_a \mathcal{A}_2 = \bigcup_{\varphi \Rightarrow \varphi' \in \mathcal{A}_1} \{\langle h_1'(\varphi), y\rangle \Rightarrow \langle h_1'(\varphi'), y\rangle\} \cup \bigcup_{\varphi \Rightarrow \varphi' \in \mathcal{A}_2} \{\langle x, h_2'(\varphi)\rangle \Rightarrow \langle x, h_2'(\varphi')\rangle\}$$

- $$\mathcal{A}_1 \otimes_p \mathcal{A}_2 = \bigcup_{\substack{\varphi_1 \Rightarrow \varphi_1' \in \mathcal{A}_1 \\ \varphi_2 \Rightarrow \varphi_2' \in \mathcal{A}_2}} \{\langle h_1'(\varphi_1), h_2'(\varphi_2)\rangle \Rightarrow \langle h_1'(\varphi_1'), h_2'(\varphi_2')\rangle$$

- $$A_1 \otimes A_2 = A_1 \otimes_a A_2 \cup A_1 \otimes_p A_2.$$

The Plan

1. Background
2. Matching Logic
3. Reachability Logic
4. Language Aggregation
5. **Proving Program Equivalence**

Proving Partial/Total Equivalence

```
s := 0
i := 1
while i <= n do
  s := s + i
  i := i + 1
return s
```

P

```
let f n a =
  if n = 0 then a
  else f (n - 1) (a + n)
in
  f n 0
```

Q

$$\left\langle P, Q \right\rangle \Rightarrow \left\langle \langle \mathtt{skip}, \mathtt{s} \mapsto x \rangle, \langle x \rangle \right\rangle$$

Proving Partial/Total Equivalence

How to prove partial/total equivalence of P and Q?

1. Construct aggregated language (syntax, model, semantics);
2. Construct pair program $\langle P, Q \rangle$;
3. Show (e.g. using reachability logic) that $\langle P, Q \rangle$ reduces (partially/totally) to desired configuration;
4. Conclude that P and Q are partially/totally equivalent.

[SYNASC 2014]

Proving Full Equivalence

Two programs are fully equivalent if:

- both terminate on the same set of inputs and
- they produce the same result for the same input.

It seems that full equivalence cannot be reduced to partial correctness or total correctness.

Proving Full Equivalence ([ICFEM 2014], [FAOC 2016])

$$\textsc{Axiom}\ \frac{\varphi \in E}{\vdash \varphi \Downarrow^{\infty} E} \textit{ and}$$

$$\textsc{Conseq}\ \frac{\models \varphi \rightarrow \exists \tilde{x}.\varphi' \qquad \vdash \varphi' \Downarrow^{\infty} E}{\vdash \varphi \Downarrow^{\infty} E} \textit{ and}$$

$$\textsc{Case Analysis}\ \frac{\vdash \varphi \Downarrow^{\infty} E \qquad \vdash \varphi' \Downarrow^{\infty} E}{\vdash \varphi \vee \varphi' \Downarrow^{\infty} E} \textit{ and}$$

$$\textsc{Step}\ \frac{\models \varphi_1 \Rightarrow_1^{*} \varphi_1' \qquad \models \varphi_2 \Rightarrow_2^{*} \varphi_2' \qquad \vdash \langle \varphi_1', \varphi_2' \rangle \Downarrow^{\infty} E}{\vdash \langle \varphi_1, \varphi_2 \rangle \Downarrow^{\infty} E} \textit{ and}$$

$$\textsc{Circ}\ \frac{\models \varphi_1 \Rightarrow_1^{+} \varphi_1' \qquad \models \varphi_2 \Rightarrow_2^{+} \varphi_2' \qquad \vdash \langle \varphi_1', \varphi_2' \rangle \Downarrow^{\infty} E \cup \{\langle \varphi_1, \varphi_2 \rangle\}}{\vdash \langle \varphi_1, \varphi_2 \rangle \Downarrow^{\infty} E}$$

Proving Full Equivalence

```
 PGM₁  ≡  c := n; n := 1; LOOP₁
LOOP₁  ≡  while (c != 1)
            n := n + 1;
            if (c % 2 != 0)
              then c := 3 * c + 1
              else c := c / 2
```

```
 PGM₂  ≡  μf.λn.λa.LOOP₂
LOOP₂  ≡  if (n != 1)
             then if (n % 2 != 0)
              then f (3 * n + 1) (a + 1)
              else f (n / 2) (a + 1)
             else a
```

$$6, 3, 10, 5, 16, 8, 4, 2, 1$$

Proving Full Equivalence

$$\varphi := n \;\texttt{!=}_{Int}\; 1 \wedge \langle\langle \texttt{LOOP}_1\texttt{,}\;\; \texttt{n} \mapsto i, \texttt{c} \mapsto n\rangle, \langle \texttt{PGM}_2\; n\; i\rangle\rangle$$

1.	$\vdash \langle\langle \texttt{skip,}\;\; \texttt{n} \mapsto i, \_\rangle, \langle i\rangle\rangle$	$\Downarrow^{\infty}$	E
2.	$\vdash \langle\langle \texttt{skip,}\;\; \texttt{n} \mapsto i, \_\rangle, \langle i\rangle\rangle$	$\Downarrow^{\infty}$	$E \cup \{\varphi\}$
3.	$\vdash \langle\langle \texttt{skip,}\;\; \texttt{n} \mapsto i, \texttt{c} \mapsto 1\rangle, \langle i\rangle\rangle$	$\Downarrow^{\infty}$	E
4.	$\vdash \langle\langle \texttt{skip,}\;\; \texttt{n} \mapsto i, \texttt{c} \mapsto 1\rangle, \langle i\rangle\rangle$	$\Downarrow^{\infty}$	$E \cup \{\varphi\}$
5.	$\vdash n \;\texttt{=}_{Int}\; 1 \wedge \langle\langle \texttt{LOOP}_1\texttt{,}\;\; \texttt{n} \mapsto i, \texttt{c} \mapsto n\rangle, \langle \texttt{PGM}_2\; n\; i\rangle\rangle$	$\Downarrow^{\infty}$	E
6.	$\vdash n \;\texttt{=}_{Int}\; 1 \wedge \langle\langle \texttt{LOOP}_1\texttt{,}\;\; \texttt{n} \mapsto i, \texttt{c} \mapsto n\rangle, \langle \texttt{PGM}_2\; n\; i\rangle\rangle$	$\Downarrow^{\infty}$	$E \cup \{\varphi\}$
7.	$\vdash n \;\texttt{!=}_{Int}\; 1 \wedge \langle\langle \texttt{LOOP}_1\texttt{,}\;\; \texttt{n} \mapsto i, \texttt{c} \mapsto n\rangle, \langle \texttt{PGM}_2\; n\; i\rangle\rangle$	$\Downarrow^{\infty}$	$E \cup \{\varphi\}$
8.	$\vdash \langle\langle \texttt{LOOP}_1\texttt{,}\;\; \texttt{n} \mapsto i, \texttt{c} \mapsto n\rangle, \langle \texttt{PGM}_2\; n\; i\rangle)$	$\Downarrow^{\infty}$	$E \cup \{\varphi\}$
9.	$\vdash n \;\texttt{!=}_{Int}\; 1 \wedge \langle\langle \texttt{LOOP}_1\texttt{,}\;\; \texttt{n} \mapsto i, \texttt{c} \mapsto n\rangle, \langle \texttt{PGM}_2\; n\; i\rangle\rangle$	$\Downarrow^{\infty}$	E
10.	$\vdash \langle\langle \texttt{LOOP}_1\texttt{,}\;\; \texttt{n} \mapsto i, \texttt{c} \mapsto n\rangle, \langle \texttt{PGM}_2\; n\; i\rangle\rangle$	$\Downarrow^{\infty}$	E
11.	$\vdash \langle\langle \texttt{LOOP}_1\texttt{,}\;\; \texttt{n} \mapsto 1, \texttt{c} \mapsto n\rangle, \langle \texttt{PGM}_2\; n\; 1\rangle\rangle$	$\Downarrow^{\infty}$	E
12.	$\vdash \langle\langle \texttt{PGM}_1\texttt{,}\;\; \texttt{n} \mapsto n\rangle, \langle \texttt{PGM}_2\; n\; 1\rangle\rangle$	$\Downarrow^{\infty}$	E

Conclusion

- Matching Logic/Reachability Logic enable reasoning about programs in a language-independent manner.
- Language Aggregation can be used to reduce partial/total equivalence to partial/full correctness.
- For full equivalence, a special proof system seems to be needed.

Future work:

- implementation
- other equivalences
- compiler correctness

Thank you!

`stefan.ciobaca@gmail.com`
`profs.info.uaic.ro/~stefan.ciobaca/`

Bibliography

- K Framework: `www.kframework.org`
- Stefan Ciobaca, Dorel Lucanu, Vlad Rusu, Grigore Rosu: **A Theoretical Foundation for Programming Languages Aggregation**. WADT 2014.
- Stefan Ciobaca: **Reducing Partial Equivalence to Partial Correctness**. SYNASC 2014.
- Stefan Ciobaca, Dorel Lucanu, Vlad Rusu, Grigore Rosu: **A Language-Independent Proof System for Mutual Program Equivalence**. ICFEM 2014.
- Stefan Ciobaca, Dorel Lucanu, Vlad Rusu, Grigore Rosu: **A Language-Independent Proof System for Full Program Equivalence**. FAOC 2016.
- Grigore Rosu, Andrei Stefanescu, Stefan Ciobaca, Brandon M. Moore: One-Path Reachability Logic. LICS 2013.
- Andrei Stefanescu, Stefan Ciobaca, Radu Mereuta, Brandon M. Moore, Traian-Florin Serbanuta, Grigore Rosu: All-Path Reachability Logic. RTA-TLCA 2014.
- Traian-Florin Serbanuta, Grigore Rosu, José Meseguer: A rewriting logic approach to operational semantics. Inf. Comput. 207(2): 305-340 (2009)
- Grigore Rosu: Matching Logic. RTA 2015.