Proving Correctness of Graph Programs Relative to Recursively Nested Conditions

Nils Erik Flick

Universität Oldenburg

February 2017
1 Correctness and Graph Programs
   - Verification Framework
   - Graph Programs
   - Recursively Nested Conditions

2 Results
   - Weakest Precondition Calculus
   - Proof Calculus
   - Expressive Power

3 Related Concepts
Aim of program verification: development of correct systems by establishing program correctness via logical deduction.

Partial correctness: whenever $P$ is run from a state satisfying $c$, if $P$ terminates then the resulting state satisfies $d$. 

Proving correctness of a program $P$ under a specification $(c, d)$. $c$ and $d$ specify state properties.
Proving correctness of a program $P$ under a specification $(c, d)$.

Checking **correctness**: compute the **weakest precondition** $W_{pP}(d)$: to postcondition $d$, this assigns precondition such that

- $P$ is correct with respect to $(W_{pP}(d), d)$
- Any $c'$ such that $P$ is correct wrt. $(c', d)$ implies $W_{pP}(d)$

Then check whether $c \Rightarrow W_{pP}(d)$. 
Graph programs are imperative programs that operate on graphs, for example:

\[
\text{Sel}(\emptyset \leftrightarrow \bullet \rightarrow \bullet) \; ; \; \text{Del}(\bullet \rightarrow \bullet \leftarrow \bullet \bullet) \; ; \\
\text{Add}(\bullet \bullet \rightarrow \bullet \rightarrow \bullet) \; ; \; \text{Uns}(\bullet \rightarrow \bullet \leftarrow \emptyset)
\]

Elementary programs: select, unselect, add, delete.
Composition: disjunction, sequence, iteration.
We want to prove correctness of graph programs relative to specifications \((c, d)\). **Nested graph conditions** are expressions like this:

\[
\forall \left(\circ \rightarrow, \exists \left(\circ \rightarrow \circ\right) \lor \exists \left(\circ \rightarrow \circ\right)\right)
\]

Unavoidable theoretical limitations:

Implication of nested conditions \((c \Rightarrow c')\) is **undecidable**.

Weakest precondition for iteration requires **invariant** finding, which cannot be fully automatic nor complete.

But in practice, verification is often possible.
Extending Nested Conditions

Many properties of interest cannot be expressed by nested conditions, for example:

- Connectedness
- Absence of cycles
- Chains of even, odd or equal length
- Chains of length $4^n$ (of theoretical interest)
- Balancedness of binary trees (useful!)

Recursively nested conditions ($\mu$-conditions) are nested conditions with **recursive** specifications.

Recursively nested conditions can express all of the above.
Recursively Nested Conditions

Example of a $\mu$-condition:

$$\forall(\circ \circ, \text{path}(\circ \circ) \Rightarrow \exists(\circ \circ, \text{paths}(\circ \circ)))$$

$$\text{path}(\circ \circ) = \exists(\rightarrow) \lor \exists(\circ \circ, \text{path}(\circ \circ))$$

$$\text{paths}(\circ \circ) = \exists(\rightarrow) \lor \exists(\circ \circ, \text{paths'}(\circ \circ))$$

$$\text{paths'}(\circ \circ) = \exists(\rightarrow) \lor \exists(\circ \circ, \text{paths'}(\circ \circ))$$
**Theorem:** the weakest precondition of a $\mu$-condition relative to an iteration-free program is again a $\mu$-condition, which can be computed.

In other words, there is a sound construction for weakest preconditions, defined for all iteration-free programs.

**Method:** a construction which transforms a finite $\mu$-condition into a finite $\mu$-condition. Soundness is proven with respect to the semantics.

**Significance:** the weakest precondition calculus is the core of the verification framework.
### The Proof Calculus $\mathcal{K}_\mu$ (I)

**Theorem**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premises</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\exists (a, c) \land d$</td>
<td>$\exists (a, c \land \exists^{-1}(a, d))$</td>
</tr>
<tr>
<td>(Supporting) Lift</td>
<td></td>
</tr>
<tr>
<td>$\neg \exists (a)$</td>
<td>$\exists (b, d)$</td>
</tr>
<tr>
<td>$\neg \exists (m^*)$</td>
<td></td>
</tr>
</tbody>
</table>

If $\exists m \in \mathcal{M}$, $m \circ b = a$ and $(m^*, b^*)$ is $\mathcal{M}$-pushout complement of $(b, m)$, $d \neq \bot$

**Partial Resolve**

$\mathcal{K}$ [Pennemann, 2009] (adapted); structural & logical rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premises</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma \vdash \Delta$</td>
<td>$D, \Gamma \vdash \Delta$</td>
</tr>
<tr>
<td>Thinning</td>
<td></td>
</tr>
<tr>
<td>$\Gamma \vdash \Theta, A$</td>
<td>$\Gamma \vdash \Theta, B$</td>
</tr>
<tr>
<td>$\Gamma \vdash \Theta, A \land B$</td>
<td>$\Gamma \vdash \Theta, A \land B$</td>
</tr>
<tr>
<td>UES</td>
<td></td>
</tr>
<tr>
<td>$\Delta, D, E, \Gamma \vdash \Theta$</td>
<td>$\Delta, E, D, \Gamma \vdash \Theta$</td>
</tr>
<tr>
<td>Contraction</td>
<td></td>
</tr>
<tr>
<td>$\Delta, E, D, \Gamma \vdash \Theta$</td>
<td>$\Delta, E, D, \Gamma \vdash \Theta$</td>
</tr>
<tr>
<td>UEA</td>
<td></td>
</tr>
<tr>
<td>$A, \Gamma \vdash \Theta$</td>
<td>$B, \Gamma \vdash \Theta$</td>
</tr>
<tr>
<td>$A \land B, \Gamma \vdash \Theta$</td>
<td>$A \land B, \Gamma \vdash \Theta$</td>
</tr>
<tr>
<td>OEA</td>
<td></td>
</tr>
<tr>
<td>$\Gamma \vdash \Theta, D$</td>
<td>$D, \Delta, \vdash \Lambda$</td>
</tr>
<tr>
<td>Interchange</td>
<td></td>
</tr>
<tr>
<td>$\Gamma \vdash \Theta, D$</td>
<td>$D, \Delta, \vdash \Lambda$</td>
</tr>
<tr>
<td>$\Gamma, \Delta \vdash \Theta, \Lambda$</td>
<td>$\Gamma, \Delta \vdash \Theta, \Lambda$</td>
</tr>
<tr>
<td>Cut</td>
<td></td>
</tr>
<tr>
<td>$\Gamma \vdash \Theta, A$</td>
<td>$\Gamma \vdash \Theta, A \lor B$</td>
</tr>
<tr>
<td>$\Gamma \vdash \Theta, A \lor B$</td>
<td>$\Gamma \vdash \Theta, A \lor B$</td>
</tr>
<tr>
<td>OES</td>
<td></td>
</tr>
</tbody>
</table>
### The Proof Calculus $\mathcal{K}_\mu$ (II)

#### Rules for handling variables and recursion:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{F} : c \vdash c'$ (resp. $c' \vdash c$)</td>
<td>(CTX)</td>
</tr>
<tr>
<td>$\mathcal{F} \uplus \mathcal{F}' : Ctx[\mathbf{x}/c] \vdash Ctx[\mathbf{x}/c']$ if $Ctx$ is monotonic (antitonic) in $\mathbf{x}$</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{F} : \Gamma \vdash \Delta, x_i^{(n)}$</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{F} : \Gamma \vdash \Delta, \mathcal{F}(\vec{x}^{(n-1)})$ $\mathcal{F}(\vec{x})$ is the right hand side for $x_i$ in $\mathcal{F}$</td>
<td>(UNROLL$_1$)</td>
</tr>
<tr>
<td>$\forall i \in I. \mathcal{H}_i(\vec{x}^{(n)}) \vdash \mathcal{G}(\mathcal{H}(\vec{x}^{(n')}))$ $\mathcal{G}(\perp) = \perp$</td>
<td>(EMPTY)</td>
</tr>
<tr>
<td>$\forall i \in I. \mathcal{H}_i(\vec{x}) = \perp$ $\vec{n}' &lt; \vec{n}$; $\mathcal{G}$ monotonic.</td>
<td></td>
</tr>
</tbody>
</table>

Further structural rules for morphism and nesting manipulation:

- $\exists (a \circ a', c)$, $\exists (a, \iota \circ \iota', c)$ and vice versa, $\exists (id, id, c)$, $\exists^{-1}(\iota, c)$, $\exists (a, c)$.
Soundness of $\mathcal{K}_\mu$

**Theorem**: the proof calculus $\mathcal{K}_\mu$ for refutation of $\mu$-conditions is sound.

**Method**: extension of the resolution-like calculus $\mathcal{K}$ by a well-founded induction rule.

**Significance**: this is the “prover” part of the verification framework. The proof calculus allows the verification of programs by attempting to prove the implication $c \Rightarrow \text{Wp}_P(d)$. 
**Theorem:** the expressiveness of $\mu$-conditions is the same as first order least fixed point logic, properly extends nested conditions and is incomparable to other known formalisms.

<table>
<thead>
<tr>
<th></th>
<th>HR</th>
<th>MSO</th>
<th>FO+lfp</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\approx$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend: $\approx$ – incomparable; $=$ – equal.

**Method:** by showing the inexpressibility of counterexamples; by translation from and to fixed point logic.

**Significance:** $\mu$-conditions are distinct from other formalisms and describe polynomial-time checkable properties.
Abstract model checking:
temporal logic specification, reduction to finite state space by
suitable state abstractions.

[Gadducci et al., 1998]
[Baldan et al., 2003]
[König and Kozioura, 2006]
[Rensink and Distefano, 2006]

This notion of correctness differs considerably from ours and no
direct comparison was attempted.
# Related Concepts: Proof-Based Approaches

<table>
<thead>
<tr>
<th>reference</th>
<th>(1)</th>
<th>(here)</th>
<th>(2)</th>
<th>(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>conditions</td>
<td>Nested</td>
<td>$\mu$-</td>
<td>HR*</td>
<td>MSO</td>
</tr>
<tr>
<td>wlp</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>proof calculus</td>
<td>complete</td>
<td>yes</td>
<td>future work</td>
<td>Hoare logic</td>
</tr>
<tr>
<td>theorem prover</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1): [Pennemann, 2009]
(2): [Radke, 2016]
(3): [Poskitt and Plump, 2014]

git:
//omega.informatik.uni-oldenburg.de/wptk.git
Conclusion

Goals achieved:

- Dijkstra-style verification approach for non-local conditions:
  - Specification language ($\mu$-conditions)
  - Weakest precondition calculus
  - Proof calculus
- Correctness under adverse conditions (2-player programs)
- Specialized results on structure-changing workflow nets (not in this talk)

Future work:

- Semi-automated prover
- Investigate proof calculus:
  - Simplification
  - Completeness
*Verification of sequential and concurrent programs*. Springer.

A logic for analyzing abstractions of graph transformation systems.  

*A discipline of programming*.  
Prentice Hall.

A fully abstract model for graph-interpreted temporal logic.  

Correctness of high-level transformation systems relative to nested conditions.  

Weakest preconditions for high-level programs.  


\[ \mathcal{F} : x^n_1 \land \neg x^n_2 \vdash \mathcal{F}_1(\bar{x}^{(n-1)}) \land \neg \mathcal{F}_2(\bar{x}^{(n-1)}) \quad \mathcal{H}_{1,2}(\bar{x}) = x_1 \land \neg x_2 \quad (1) \]

\[ \mathcal{F} : x^n_1 \land \neg x^n_2 \vdash (\exists(1 \leftarrow 2) \lor \exists(1(3) \leftarrow 2(2), x_1^{(n-1)}[1 \leftarrow 2]) \land \neg \exists(1 \leftarrow 2) \land \neg \exists(1(3) \leftarrow 2(2), x_2^{(n-1)}[1 \leftarrow 2]) \quad (2) \]

\[ \mathcal{F}' : \ldots \vdash \exists(1 \leftarrow 2, x_1^{(n-1)}[1 \leftarrow 2]) \land \neg \exists(1(3) \leftarrow 2(2), x_2^{(n-1)}[1 \leftarrow 2]) \quad (3) \]

\[ \mathcal{F}' : x^n_1 \land \neg x^n_2 \vdash \exists(1 \leftarrow 2, x_1^{(n-1)}[1 \leftarrow 2] \land \neg x_2^{(n-1)}[1 \leftarrow 2]) \quad (3) \]

\[ \mathcal{F} : x_1 \land \neg x_2 \vdash \bot \]
Adversity: the Role of Nondeterminism

The semantics $\semantics{P}$ assigns to $P$ the set of all possible pairings (input, output) that correspond to executions of $P$.

Graph programs are nondeterministic:
1) There may be several ways to make a selection.
2) **Disjunctive** composition: $\semantics{P + Q} = \semantics{P} \cup \semantics{Q}$.
3) **Loops** may be executed arbitrarily often.

Addition, deletion and unselection are deterministic.

The weakest precondition transformation takes nondeterminism into account.
Operational **semantics** is introduced and related to $\llbracket P \rrbracket$: intermediary states appear as (current graph, remaining program).

**A model of adversity:**

Each intermediary state belongs **either** to **sys** (+) or to **env** (−).

Difference between system and environment lies in the **treatment of nondeterminism**.

Semantics $\llbracket P \rrbracket$ is the same but can be restricted ($\llbracket P \rrbracket_\chi$) by a choice function $\chi : (+)-states \rightarrow successor\ states$. 
In the definition of the weakest precondition construction:

Nondeterminism resolved by `sys` has existential quantifiers / disjunction where nondeterminism resolved by `env` has universal ones / conjunction.

Otherwise, the framework did not need to be modified.

Soundness of the newly defined weakest precondition was checked against the operational semantics, which in turn is equivalent to the “denotational” semantics.
**Theorem**: the extended weakest precondition construction for two-player programs (with \textit{sys} and \textit{env}-constructors) is sound.

This result on \textbf{system correctness under adverse conditions} holds for \textbf{\( \mu \)-conditions}, for which the weakest precondition was first proven in the one-player case.

The classical situation already models \textbf{adversity}, but nothing else. The new part is the interaction of \((+/−)\)-nondeterminism.
Adversity: Future Work

Definition of **parallel composition** to be used within the same framework. This poses additional problems such as $(+/−)$-race conditions, but could be worthwhile for modeling.

**Controller synthesis.** Knowing that a choice function $\chi$ exists is distinct from actually obtaining such a function.