Verification of Graph Programs

ICGT-DS 2012, Universität Bremen

Chris Poskitt

Department of Computer Science

The University of York

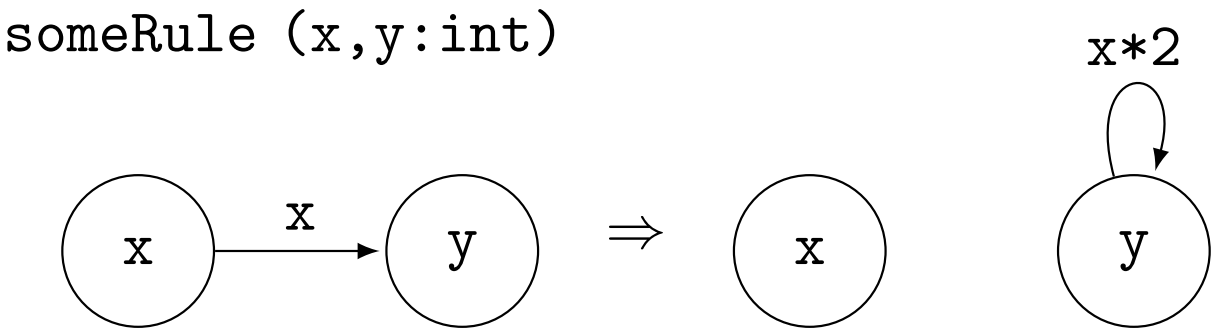
26th September 2012

Graph Programs (GP)

- program states are graphs labelled by (sequences of) integers and strings
- graphs manipulated directly by rule schemata
- rule schemata applied according to simple control constructs
  - sequential composition, conditionals, iteration, ...
- formal operational semantics; $[\![P]\!]G$ yields a set of graphs (possibly also $\perp$)

Rule schemata in GP

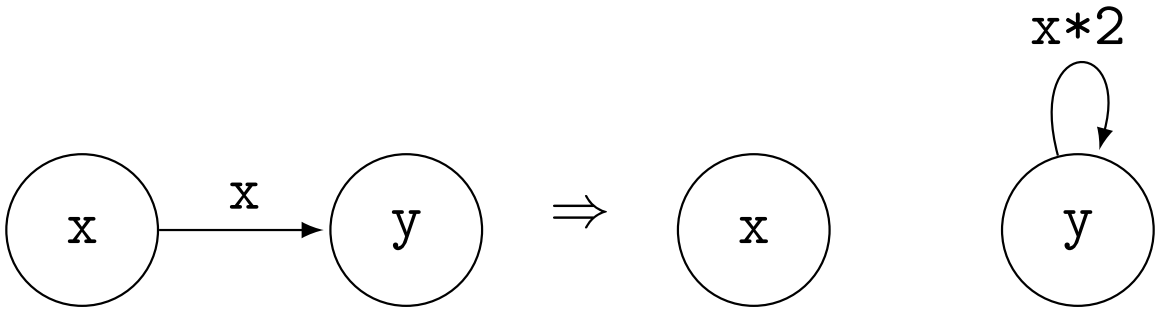
- generalise DPO rules with expressions and relabelling



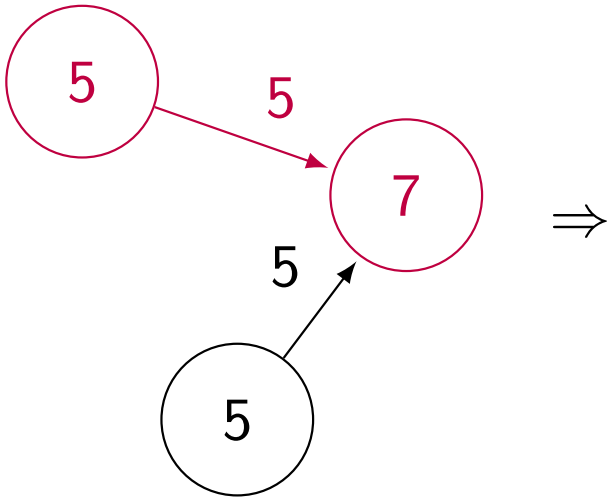
- a variable assignment is found in the graph matching process

- then expressions evaluated to yield rule with concrete labels

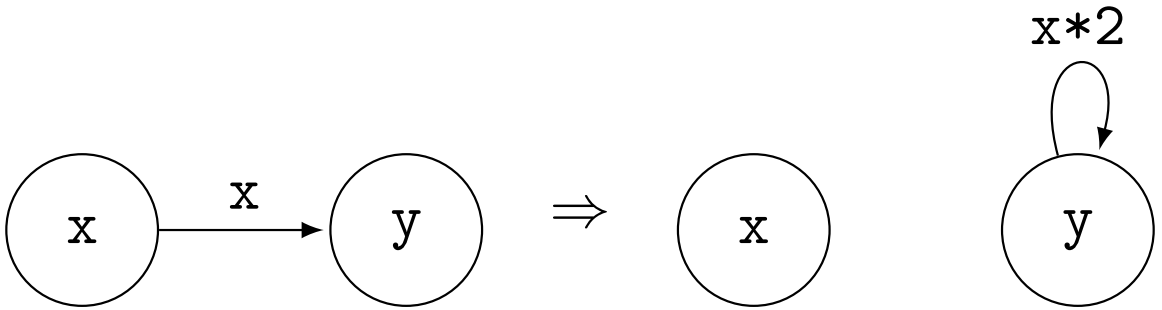
Rule application



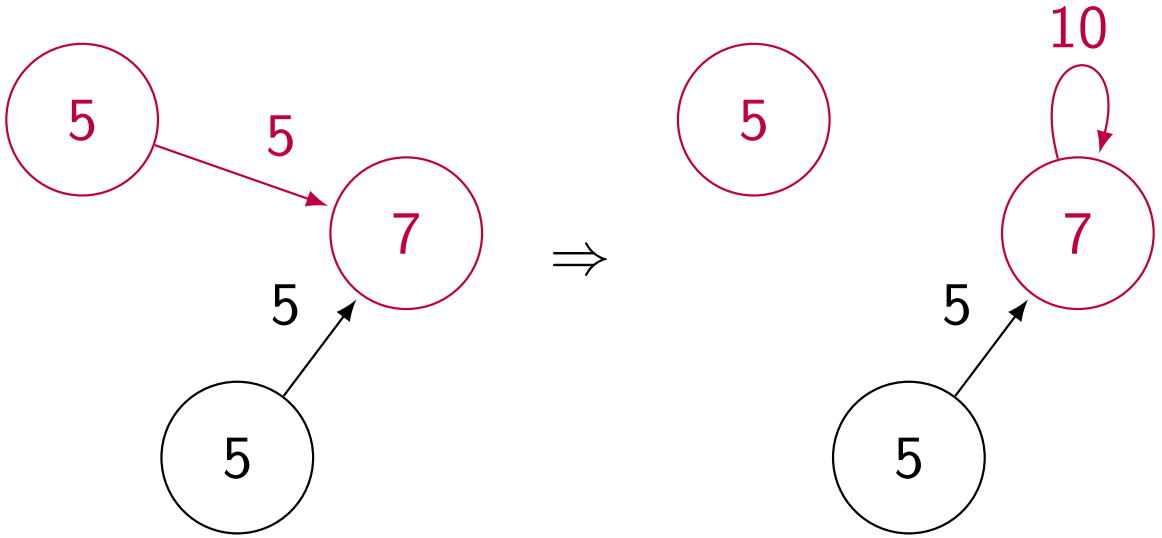
is applied after the instantiation of variables $\alpha = (\mathrm{x} \mapsto 5, \mathrm{y} \mapsto 7)$



Rule application



is applied after the instantiation of variables $\alpha = (\mathrm{x} \mapsto 5, \mathrm{y} \mapsto 7)$



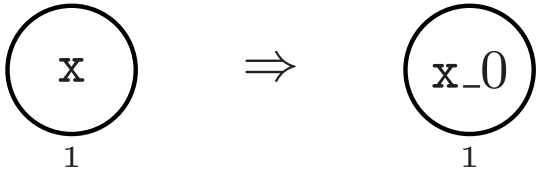
Graph program example

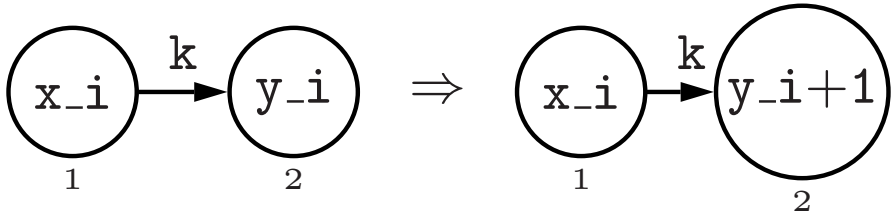
. . . computing a colouring

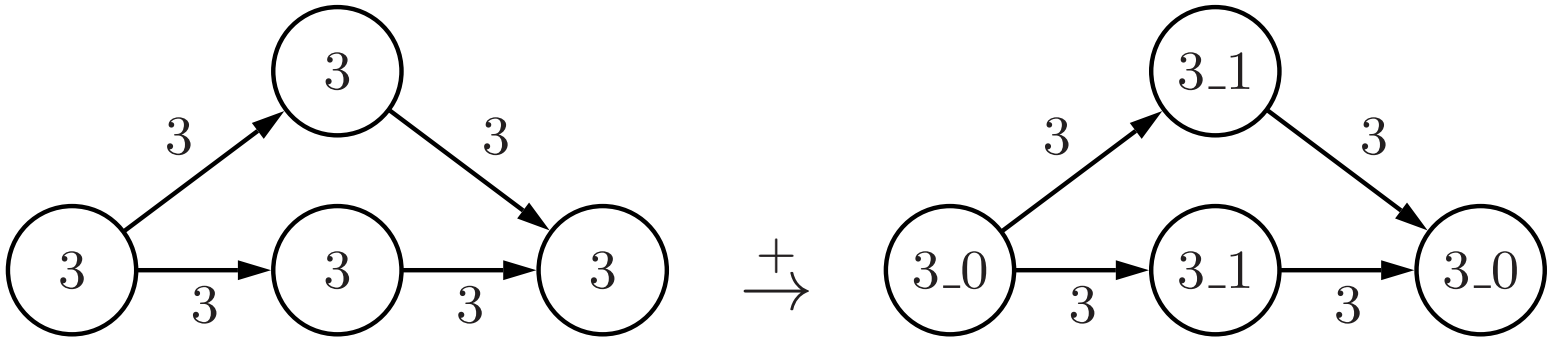
```
main = init!; inc!

init(x: int)

inc(i, k, x, y: int)
```







What about correctness?

. . . in the setting of graphs!

Was that program "correct"? What does that mean?

According to what specification?

What do pre- and postconditions look like?

How do you compute a weakest precondition?

Our approach

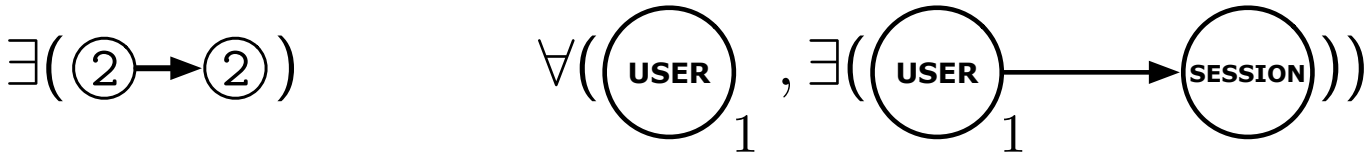
. . . adapting Hoare-style reasoning to the graph programming world!

Thesis contributions:

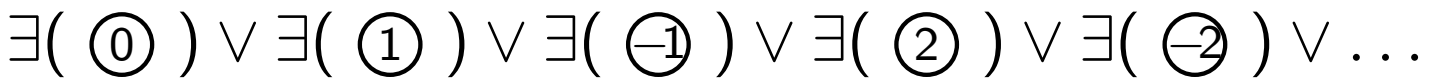
- nested conditions with expressions (or E-conditions)
- many-sorted logic for graphs, with hierarchical types
- Hoare calculi for partial and total correctness proofs
- case studies: garbage collection, pointers

Assertion language for graphs

- nested conditions: graphical, morphism-based formalism for FO structural properties



- but insufficient for graph programs. Consider:

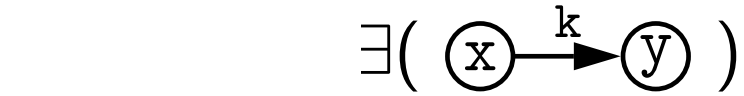


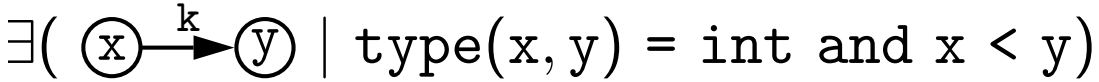
E-conditions

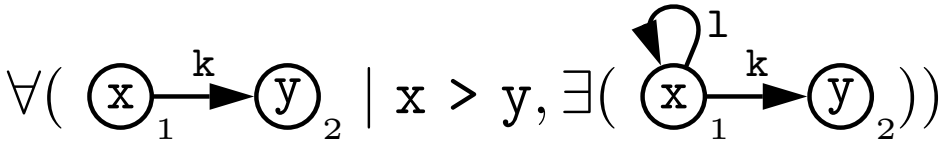
- add expressions and assignment constraints to yield E-conditions

$$\exists(\ \textcircled{x}\ \mid \mathrm{type}(x) = \mathrm{int})$$

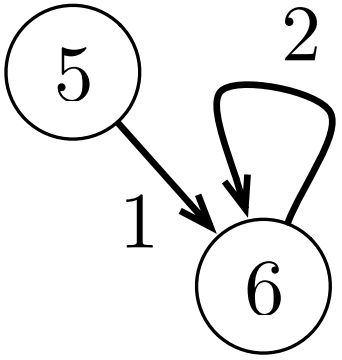
- graphically and precisely reason about both structure and relations between labels





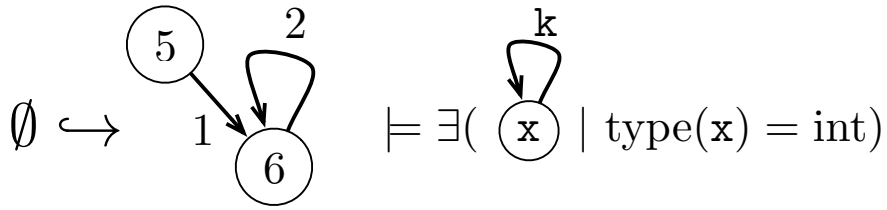


Satisfaction relation $\models$

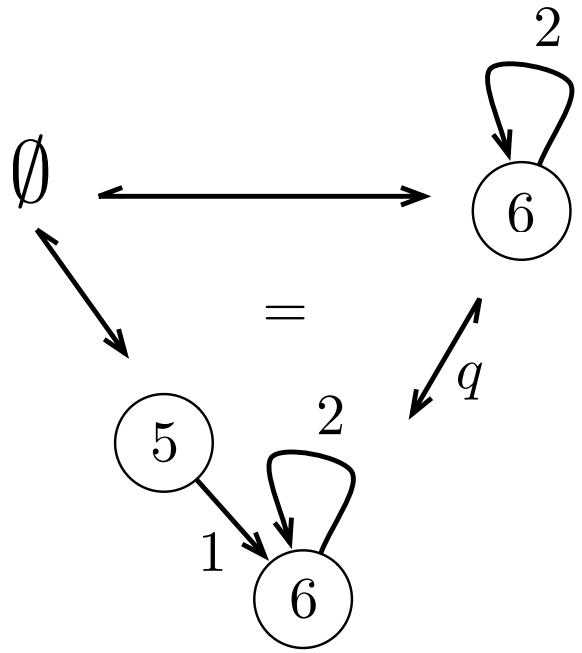


$$\models \exists(\ \overset{\mathtt{k}}{\underset{\mathtt{x}}{\bigcirc}}\ \mid \mathrm{type}(\mathbf{x}) = \mathrm{int})$$

Satisfaction relation $\models$



By assignment $\alpha = (\mathtt{x} \mapsto 6, \mathtt{k} \mapsto 2)$.



Computing a precondition

- transformation Pre from rule schema and postcondition to a precondition, i.e.

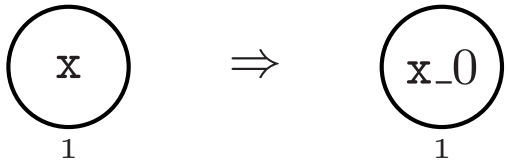
$$\models_{\text{par}} \ \{\text{Pre}(r, post)\}\ r\ \{post\}$$

- based on shifting lemmata for nested conditions
- but also with label unification
- idea of Pre "how might r cause $post$ to no longer hold (or cause it to start holding); how do they interact?"

Example

Rule schema:

$$\texttt{init(x: int)}$$



Postcondition:

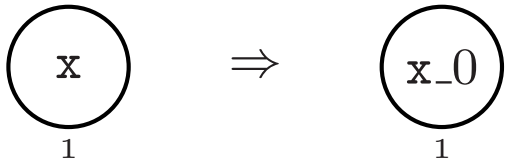
$$e \quad = \quad \forall(\ \textcircled{a}_1, \exists(\ \textcircled{a}_1 \mid \texttt{type(a) = int}) \vee \exists(\ \textcircled{a}_1 \mid \texttt{a = b\_c and type(b, c) = int}))$$

Precondition computed. . .

Example

Rule schema:

$$\texttt{init(x: int)}$$



Postcondition:

$$e \quad = \quad \forall(\,\textcircled{a}_1, \exists(\,\textcircled{a}_1 \mid \texttt{type(a) = int}) \vee \exists(\,\textcircled{a}_1 \mid \texttt{a = b\_c and type(b, c) = int}))$$

Precondition computed...

$$\begin{aligned}\mathrm{Pre}(\texttt{init}, e) \quad = \quad & \forall(\,\textcircled{x}_1 \mid \texttt{type(x) = int},\\ & \forall(\,\textcircled{x}_1\,\textcircled{a}_2, \exists(\,\textcircled{x}_1\,\textcircled{a}_2 \mid \texttt{type(a) = int})\\ & \quad \vee \exists(\,\textcircled{x}_1\,\textcircled{a}_2 \mid \texttt{a = b\_c and type(b, c) = int}))\\ & \wedge\forall(\,\textcircled{x}_1, \exists(\,\textcircled{x}_1 \mid \texttt{type(x\_0) = int})\\ & \quad \vee\exists(\,\textcircled{x}_1 \mid \texttt{x\_0 = b\_c and type(b, c) = int})))\end{aligned}$$

A many-sorted logic

- variable sorts (types) for nodes, edges (like Courcelle) but also labels (infinite semantic domain)

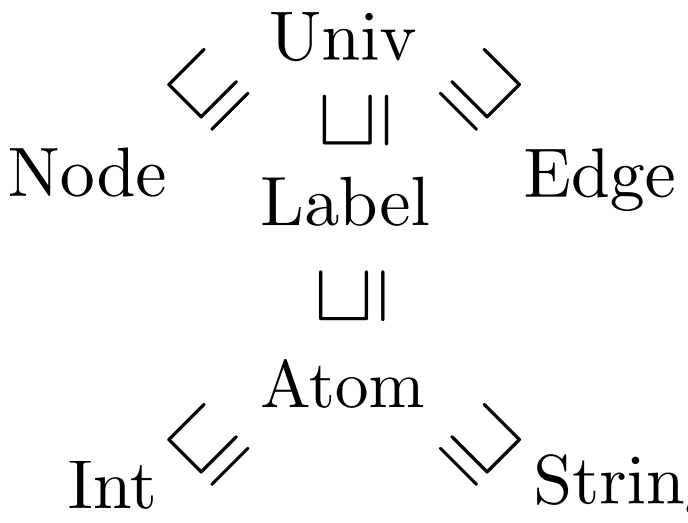
$$\exists \texttt{e:E}.\ \exists \texttt{x:I}.\ \texttt{label(e)} = \texttt{x} \wedge \texttt{x} < 0$$

- avoids morphisms and nesting

- translations from E-conditions to many-sorted formulae (and back)

MS logic – more advantages

- reduced search space for deduction procedures
- hierarchy of types (upper semilattice)



  e.g. $e_1 > e_2$ [integer labels only] vs. $e_1 = e_2$ [any labels]
- nodes/edges easily identified
  - $\forall$`x,y:V. 5 > 0` $\rightarrow$ `x = y`
- understand expressiveness of E-conditions

Hoare-style reasoning

$$\models_{\mathsf{par}} \{c\}\ P\ \{d\}$$

- **partial correctness:** if program P is executed on a graph satisfying c, then any graph resulting satisfies d

$$\models_{\mathsf{wtot}} \{c\}\ P\ \{d\}$$

- **weak total correctness:** partial correctness, but also the guarantee that P eventually terminates

$$\models_{\mathsf{tot}} \{c\}\ P\ \{d\}$$

- **total correctness:** weak total correctness, but also the guarantee that no execution reaches a failure state

Partial correctness proof rules

$$[\text{ruleapp}]\ \frac{}{\{\text{Pre}(r, c)\}\ r\ \{c\}} \qquad [\text{nonapp}]\ \frac{}{\{\neg\text{App}(\mathcal{R})\}\ \mathcal{R}\ \{\text{false}\}}$$

$$[\text{ruleset}]\ \frac{\{c\}\ r_1\ \{d\}\ \ldots\ \{c\}\ r_n\ \{d\}}{\{c\}\ \{r_1, \ldots, r_n\}\ \{d\}} \qquad [!]\ \frac{\{inv\}\ \mathcal{R}\ \{inv\}}{\{inv\}\ \mathcal{R}!\ \{inv \wedge \neg\text{App}(\mathcal{R})\}}$$

$$[\text{comp}]\ \frac{\{c\}\ P\ \{e\} \qquad \{e\}\ Q\ \{d\}}{\{c\}\ P; Q\ \{d\}} \qquad [\text{cons}]\ c \Rightarrow c'\ \frac{\{c'\}\ P\ \{d'\}}{\{c\}\ P\ \{d\}}\ d' \Rightarrow d$$

$$[\text{if}_1]\ \frac{\{c \wedge \text{App}(\mathcal{R})\}\ P\ \{d\} \qquad \{c \wedge \neg\text{App}(\mathcal{R})\}\ Q\ \{d\}}{\{c\}\ \texttt{if}\ \mathcal{R}\ \texttt{then}\ P\ \texttt{else}\ Q\ \{d\}}$$

Soundness and relative completeness

Soundness:

- proof rules are sound for partial correctness
- proof uses characterisations of Pre, App, and the operational semantics of GP

Completeness:

- are the rules complete in the extensional approach?
- with E-conditions as assertions, are the rules relatively complete?

Total correctness

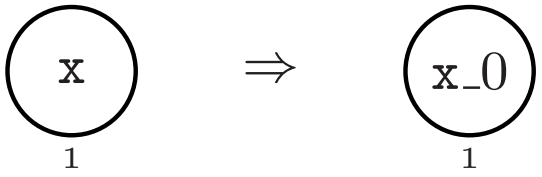
- more on Friday @ GCM 2012!
- **wtot:** show that iterated rules $\mathcal{R}!$ are decreasing for some termination function
- **tot:** show that preconditions imply rule applicability

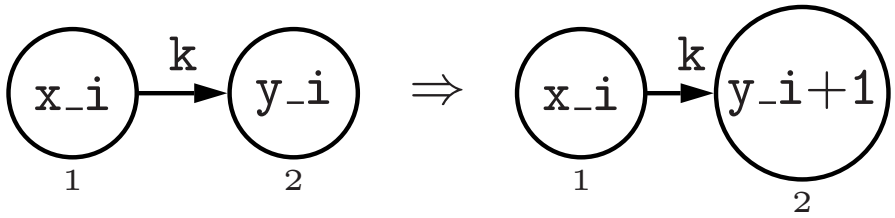
Recap: graph colouring

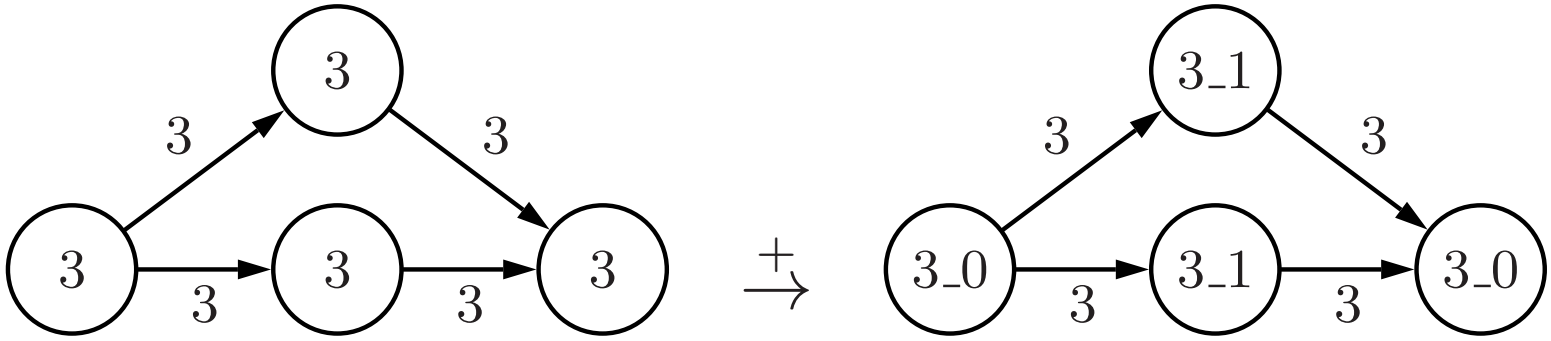
```
main = init!; inc!

init(x: int)

inc(i, k, x, y: int)
```







A partial correctness proof tree

. . . (total correctness proofs on Friday!)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{\mathrm{Pre}(\texttt{init}, e)\}\ \texttt{init}\ \{e\}}[\text{ruleapp}]}{\{e\}\ \texttt{init}\ \{e\}}[\text{cons}]}{\{e\}\ \texttt{init!}\ \{e \wedge \neg\mathrm{App}(\{\texttt{init}\})\}}[!]}{\{c\}\ \texttt{init!}\ \{d\}}[\text{cons}] \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\{\mathrm{Pre}(\texttt{inc}, d)\}\ \texttt{inc}\ \{d\}}[\text{ruleapp}]}{\{d\}\ \texttt{inc}\ \{d\}}[\text{cons}]}{\{d\}\ \texttt{inc!}\ \{d \wedge \neg\mathrm{App}(\{\texttt{inc}\})\}}[!]}{\{c\}\ \texttt{init!};\ \texttt{inc!}\ \{d \wedge \neg\mathrm{App}(\{\texttt{inc}\})\}}[\text{comp}]$$

$$\begin{aligned}
c &= \neg\exists(\ \textcircled{a}\ \mid \texttt{not type(a) = int}) \\
d &= \forall(\ \textcircled{a}_1, \exists(\ \textcircled{a}_1 \mid \texttt{a = b_c and type(b, c) = int})) \\
e &= \forall(\ \textcircled{a}_1, \exists(\ \textcircled{a}_1 \mid \texttt{type(a) = int}) \vee \exists(\ \textcircled{a}_1 \mid \texttt{a = b_c and type(b, c) = int})) \\
\neg\mathrm{App}(\{\texttt{init}\}) &= \neg\exists(\ \textcircled{x}\ \mid \texttt{type(x) = int}) \\
\neg\mathrm{App}(\{\texttt{inc}\}) &= \neg\exists(\ \textcircled{x_i} \xrightarrow{\texttt{k}} \textcircled{y_i}\ \mid \texttt{type(i, k, x, y) = int})
\end{aligned}$$

A partial correctness proof tree

. . . (total correctness proofs on Friday!)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{\mathrm{Pre}(\texttt{init}, e)\}\ \texttt{init}\ \{e\}}[\text{ruleapp}]}{\{e\}\ \texttt{init}\ \{e\}}[\text{cons}]}{\{e\}\ \texttt{init!}\ \{e \wedge \neg\mathrm{App}(\{\texttt{init}\})\}}[!]}{\{c\}\ \texttt{init!}\ \{d\}}[\text{cons}] \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\{\mathrm{Pre}(\texttt{inc}, d)\}\ \texttt{inc}\ \{d\}}[\text{ruleapp}]}{\{d\}\ \texttt{inc}\ \{d\}}[\text{cons}]}{\{d\}\ \texttt{inc!}\ \{d \wedge \neg\mathrm{App}(\{\texttt{inc}\})\}}[!]}{\{c\}\ \texttt{init!};\ \texttt{inc!}\ \{d \wedge \neg\mathrm{App}(\{\texttt{inc}\})\}}[\text{comp}]$$

$$\begin{aligned}
c &= \neg\exists(\ \textcircled{a}\ \mid \texttt{not type(a) = int}) \\
d &= \forall(\ \textcircled{a}_1, \exists(\ \textcircled{a}_1 \mid \texttt{a = b_c and type(b, c) = int})) \\
e &= \forall(\ \textcircled{a}_1, \exists(\ \textcircled{a}_1 \mid \texttt{type(a) = int}) \vee \exists(\ \textcircled{a}_1 \mid \texttt{a = b_c and type(b, c) = int})) \\
\neg\mathrm{App}(\{\texttt{init}\}) &= \neg\exists(\ \textcircled{x}\ \mid \texttt{type(x) = int}) \\
\neg\mathrm{App}(\{\texttt{inc}\}) &= \neg\exists(\ \textcircled{x_i} \xrightarrow{\texttt{k}} \textcircled{y_i}\ \mid \texttt{type(i, k, x, y) = int}) \\
\mathrm{Pre}(\texttt{init}, e) &= \forall(\ \textcircled{x}_1 \mid \texttt{type(x) = int}, \\
&\quad \forall(\ \textcircled{x}_1\ \textcircled{a}_2, \exists(\ \textcircled{x}_1\ \textcircled{a}_2 \mid \texttt{type(a) = int}) \\
&\quad\quad \vee \exists(\ \textcircled{x}_1\ \textcircled{a}_2 \mid \texttt{a = b_c and type(b, c) = int})) \\
&\quad \wedge\forall(\ \textcircled{x}_1, \exists(\ \textcircled{x}_1 \mid \texttt{type(x_0) = int}) \\
&\quad\quad \vee\exists(\ \textcircled{x}_1 \mid \texttt{x_0 = b_c and type(b, c) = int})))
\end{aligned}$$

Case studies and future work

- case studies (thesis task!) – garbage collection, pointers, ...
- formalisation of the calculi in a proof assistant (a big task)
- the implication problem for E-conditions, i.e. is $c \Rightarrow d$ valid?
- stronger assertion language, e.g. hyperedge replacement (HR) conditions

Papers on this work

E-conditions and partial correctness:

- C.M. Poskitt and D. Plump. Hoare-Style Verification of Graph Programs. Fundamenta Informaticae, 118(1-2):135-175, 2012.
- C.M. Poskitt and D. Plump. A Hoare Calculus for Graph Programs. In Proc. ICGT 2010, volume 6372, pages 139-154. Springer-Verlag, 2010.

Many-sorted logic and translations:

- C.M. Poskitt, D. Plump, and A. Habel. A Many-Sorted Logic for Graph Programs. Submitted for publication.

Total correctness:

- C.M. Poskitt and D. Plump. Verifying Total Correctness of Graph Programs. In Proc. GCM 2012. See you on Friday? ☺

Thank you! Danke schön!

http://www.cs.york.ac.uk/~cposkitt