Verifying Total Correctness of Graph Programs
GCM 2012, Universität Bremen

Chris Poskitt and Detlef Plump

Department of Computer Science
THE UNIVERSITY of York

28th 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; $\llbracket P \rrbracket G$ yields a set of graphs (possibly also $\bot$)
Rule schemata in GP

- generalise DPO rules with expressions and relabelling

\[
\text{someRule} \ (x,y:\text{int})
\]

- 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 $\alpha = (x \mapsto 5, y \mapsto 7)$
Rule application

\[ x \rightarrow y \Rightarrow x\ y \rightarrow x \Rightarrow y \]

is applied after the instantiation of variables \( \alpha = (x \mapsto 5, y \mapsto 7) \)
Graph program example

... reversing edges

main = reverse!; unmark!

reverse(a, x, y: int) unmark(a, x, y: int)

x
1
y
2
a ⇒ x
1
y
2

unmark(a, x, y: int)

Fig. 6. The program inverse expressions joined by underscores. Here, we need to mark reversed edges as otherwise the loop reverse! would not terminate. Note that the rule schema reverse can only be applied to edges with untagged labels.

Example 3 (Shortest distances). Given a graph $G$ whose edge labels are integers, the distance of a directed path from a node $v$ to a node $w$ is the sum of the edge labels on that path. If all edge labels in $G$ are nonnegative, then the shortest distance from $v$ to $w$ is the minimum of the distances of all paths from $v$ to $w$.

The program distances in Figure 7 expects an integer-labelled input graph where exactly one node $v$ has a tagged label of the form $x_0$ and where all edge labels are nonnegative. It adds to each node $w$ that is distinct and reachable from $v$ the shortest distance from $v$ to $w$.

main = {add, reduce}!

add(a, b, x, y: int) x
a
b ⇒ x
a + b

reduce(a, b, c, x, y: int) x
a
y
c
b ⇒ x
a + b

where $a + b < c$
Graph program example

...reversing edges

main = reverse!; unmark!

reverse(a, x, y: int)

unmark(a, x, y: int)

x
1
y
2
a ⇒ x
1
y
2
a
0

Fig. 6. The program

Example 3 (Shortest distances). Given a graph $G$ whose edge labels are integers, the distance of a directed path from a node $v$ to a node $w$ is the sum of the edge labels on that path. If all edge labels in $G$ are nonnegative, then the shortest distance from $v$ to $w$ is the minimum of the distances of all paths from $v$ to $w$.

The program $distances$ in Figure 7 expects an integer-labelled input graph where exactly one node $v$ has a tagged label of the form $x_0$ and where all edge labels are nonnegative. It adds to each node $w$ that is distinct and reachable from $v$ with the shortest distance from $v$ to $w$.

In each iteration of the program’s loop, one of the rule schemata $add$ and $reduce$ is applied to the current graph. If both rule schemata are applicable, one of them is chosen nondeterministically. An equivalent, slightly more deterministic solution is to separate the phases of addition and reduction:

main = add!; reduce!

A refined version of the program $distances$ which implements Dijkstra’s shortest-path algorithm can be found in [19].
What about correctness?

- we are studying **Hoare-style reasoning** for graph programs

- reasoning around **individual constructs**
  - (vs. Pennemann et al. weakest precondition approach)

- proposed a calculus for proving **partial correctness**

- morphism- and logic-based assertion languages

- **today**: what about **total correctness**?
Notions of correctness

... how to define $\models \{pre\} P \{post\}$
Notions of correctness

... how to define $\models \{ pre \} P \{ post \}$

- define with respect to the formal semantics:

$$\llbracket P \rrbracket_G = \{ H \in G(\mathcal{L}) \mid \langle P, G \rangle \Downarrow H \}$$

$\cup \ldots$
Notions of correctness

... how to define $\models \{\text{pre}\} P \{\text{post}\}$

- define with respect to the formal semantics:

$$\llbracket P \rrbracket G = \{H \in G(\mathcal{L}) \mid \langle P, G \rangle \xrightarrow{\ddag} H\}$$

$$\cup \{\bot \mid P \text{ can diverge or get stuck from } G\}$$

- must account for nondeterminism and non-termination

- (“getting stuck” can occur if iterating programs contain iterating programs, but these are not handled by our Hoare calculi for technical reasons)
Partial and total (?) correctness

- **Partial correctness:**

  \[ \models_{\text{par}} \{ \text{pre} \} P \{ \text{post} \} \]

  If program \( P \) is executed on a graph satisfying \( \text{pre} \), then any graph resulting satisfies \( \text{post} \)

- **Total (?) correctness:**

  \[ \models_{\text{tot}(?)} \{ \text{pre} \} P \{ \text{post} \} \]

  Partial correctness, but also the guarantee that \( P \) eventually terminates
Partial and total (?) correctness

- **partial correctness:**

\[ \models_{\text{par}} \{ \text{pre} \} \ P \ \{ \text{post} \} \]

if program \( P \) is executed on a graph satisfying \( \text{pre} \), then any graph resulting satisfies \( \text{post} \)

- **total (?) correctness:**

\[ \models_{\text{tot}(?)} \{ \text{pre} \} \ P \ \{ \text{post} \} \]

partial correctness, but also the guarantee that \( P \) eventually terminates

- ...but will we actually yield a graph?
Failure points

... look to the semantics

- if a set of rule schemata $\mathcal{R}$ is not applicable to a graph $G$...

\[ \text{[Call$_2$]sos} \quad \frac{G \not\vdash \mathcal{R}}{\langle \mathcal{R}, G \rangle \rightarrow \text{fail}} \]

... then a failure state is reached

- if a terminating program is free of failing executions, then a graph will certainly result
Two notions of total correctness

... following Apt

- **weak total correctness:**

  \[ \models_{\text{wtot}} \{ c \} \ P \ {d} \]

  partial correctness, but also the guarantee that \( P \) eventually terminates

- **total correctness:**

  \[ \models_{\text{tot}} \{ c \} \ P \ {d} \]

  weak total correctness, but also the guarantee that no execution reaches a failure state
Some partial correctness proof rules

(that are certainly not totally correct!)

- as-long-as-possible iteration

\[
\begin{array}{c}
[!] \quad \{inv\} \mathcal{R} \{inv\} \\
\{inv\} \mathcal{R}! \{inv \land \neg \text{App} (\mathcal{R})\}
\end{array}
\]

- application of a rule schemata set

\[
\begin{array}{c}
\text{[ruleset]} \quad \{c\} \mathcal{R} \{d\} \quad \ldots \quad \{c\} \mathcal{R} \{d\} \\
\{c\} \{r_1, \ldots, r_n\} \{d\}
\end{array}
\]
Addressing divergence

\[ \frac{\vdash \text{par } \{ inv \} \mathcal{R} \{ inv \}, \quad \mathcal{R} \text{ is } \#\text{-decreasing under } inv}{\{ inv \} \mathcal{R}! \{ inv \land \neg \text{App}(\mathcal{R}) \}} \]
Addressing divergence

Moreover, if an execution of $P$ does in fact terminate, we cannot be sure that it does so without failure. When referring to total correctness, we follow [1] in meaning both absence of divergence and failure; whereas referring to weak total correctness, we mean only absence of divergence.

Definition 3 (Weak total correctness).

A graph program $P$ is weakly totally correct with respect to a precondition $c$ and postcondition $d$ (both of which are $E$-constraints), denoted $\left\{ c \right\} P \left\{ d \right\}$, if

\[
\left\{ c \right\} \text{par} \left\{ c \right\} P \left\{ d \right\}
\]

and if for every graph $G \in G(L)$ such that $G \left\{ c \right\}$, there is no infinite sequence $\langle P \hookrightarrow G \rangle \rightarrow \langle P_1 \hookrightarrow G_1 \rangle \rightarrow \langle P_2 \hookrightarrow G_2 \rangle \rightarrow \cdots$.

Definition 4 (Total correctness).

A graph program $P$ is totally correct with respect to a precondition $c$ and postcondition $d$ (both of which are $E$-constraints), denoted $\left\{ c \right\} P \left\{ d \right\}$, if $\left\{ c \right\} P \left\{ d \right\}$, and for any graph $G \in G(L)$ such that $G \left\{ c \right\}$, there is no derivation $\langle P \hookrightarrow G \rangle \rightarrow^* \langle \rangle$.

Our proof system for weak total correctness is formed from the proof rules of Figure 3, but with $[!]$ replaced by $[!]_\text{tot}$ in Figure 4. If a triple $\left\{ c \right\} P \left\{ d \right\}$ can be derived from this proof system, we write $\vdash \left\{ c \right\} P \left\{ d \right\}$. The issue of termination is localised to the proof rule for as-long-as-possible iteration: $[!]_\text{tot}$ has an additional premise to $[!]$ which handles this. It requires, for a particular rule schemata set, that there is a function assigning naturals to graphs such that these naturals are decreasing along derivation steps. Such a function $\#$ is called a termination function.

These definitions are given more precisely below.

\[
\vdash \text{par} \left\{ \text{inv} \right\} R \left\{ \text{inv} \right\}, \quad R \text{ is } \#\text{-decreasing under } \text{inv}
\]

\[
\left\{ \text{inv} \right\} R! \left\{ \text{inv} \land \neg \text{App}(R) \right\}
\]

- $\#$ is a termination function: a mapping from graphs to natural numbers

- $R$ is $\#$-decreasing under $\text{inv}$ if for all graphs $G, H$ satisfying $\text{inv}$,

\[
G \Rightarrow_R H \implies \#G > \#H
\]
Example

...totally correct graph colouring

main = init!; inc!

init(x: int)

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

The program colouring and one of its executions

The program initially colours each node with 0 by applying the rule schema init as long as possible, using the iteration operator '!' . It then repeatedly increments the target colour of edges with the same colour at both ends. Note that this process is nondeterministic: Figure 1 shows one possible execution; there is another execution resulting in a graph with three colours.

The program reachable in Figure 2 checks if there is a path from one distinguished node (tagged with 1, i.e. \(x\_1\)) to another (tagged with 2, i.e. \(y\_2\)), returning the input graph if there is one, otherwise returning the same graph but with a new direct link between them. It repeatedly propagates 0-tagged nodes from the 1-tagged node (and subsequent 0-tagged nodes) for as long as possible via prop!. It then tests via reachable whether there is a direct link between the distinguished nodes, or a link from a 0-tagged node to the 2-tagged node (indicating a path). If so, nothing happens; otherwise, a direct link is added via addlink. In both cases, the 0-tags are removed by the iteration of undo.

GP's formal semantics is given in the style of structural operational semantics. Inference rules (omitted here, but given in [12]) inductively define a small-step transition relation \(\rightarrow\) on configurations. In our setting, a configuration is either a command sequence (ComSeq) together with a graph (i.e. an unfinished computation), just a graph, or the special element fail (representing a failure state). The meaning of graph programs is summarised by a semantic.
Example

... totally correct graph colouring

- \( \#_{\text{init}} \) defined to return the total number of nodes labelled by a unit-length integer sequence
- \( \#_{\text{inc}} \) is less obvious

\[
\text{inc}(i, k, x, y: \text{int})
\]

\[
\begin{align*}
x_i & \rightarrow^k y_i \\
1 & \rightarrow 2
\end{align*}
\]

\[
\begin{align*}
x_i & \rightarrow^k y_{i+1} \\
1 & \rightarrow 2
\end{align*}
\]
Example

... totally correct graph colouring

- \#_{init} defined to return the total number of nodes labelled by a unit-length integer sequence
- \#_{inc} is less obvious

\[ \text{inc}(i, k, x, y: \text{int}) \]

\[ \begin{array}{c}
\text{x}_i \xrightarrow{k} \text{y}_i \\
1 \quad 2
\end{array} \quad \Rightarrow \quad \begin{array}{c}
\text{x}_i \xrightarrow{k} \text{y}_{i+1} \\
1 \quad 2
\end{array} \]

\[ \#_{inc} G = \sum_{i=0}^{\lfloor V_G \rfloor - 1} i - \sum_{v \in V_G} \text{colour}(v) \]
Addressing failure

\[
[\text{ruleset}]_{\text{tot}} \quad \frac{c \Rightarrow \text{App}(\mathcal{R}), \quad \vdash \text{par} \{c\} \ r \ \{d\} \ \text{for each} \ r \in \mathcal{R}}{\{c\} \ \mathcal{R} \ \{d\}}
\]

- \text{App}(\mathcal{R}) \ is \ satisfied \ only \ by \ graphs \ for \ which \ \mathcal{R} \ is \ applicable \ to

- we have defined \text{App} for \text{E-conditions}, our graphical and morphism-based assertion language

- if \(c\) implies the applicability of \(\mathcal{R}\), then \(\mathcal{R}\) will not fail on graphs satisfying \(c\)
Example

...reachable?

main = prop!; (if reachable then skip else addlink); undo!

prop(a, x, y, z: int)

reachable(a, x, y, z: int)

where y=1 or y=0

addlink(x, y: int)

undo(x: int)
Example

... reachable?

- when addlink is addressed in a proof tree, the precondition must imply that addlink is applicable:

\[
\text{App}\{\text{addlink}\} = \exists (x_1 \ y_2 \mid \text{type}(x, y) = \text{int})
\]

- part of our example from the paper:

\[
\begin{align*}
\text{[ruleset]}_\text{tot} & : d \land e \land \neg \text{App}\{\text{reachable}\} \Rightarrow \text{App}\{\text{addlink}\} \\
\text{[cons]} & : \frac{\text{Pre}(\text{addlink}, d \land e) \ \text{addlink}\{d \land e\}}{\text{par}\{d \land e \land \neg \text{App}\{\text{reachable}\}\} \ \text{addlink}\{d \land e\}}
\end{align*}
\]
Technical results

... soundness, and completeness for termination

Soundness:

- $\vdash_{\text{wtot}} \{\text{pre}\} \ P \ \{\text{post}\}$ implies $\models_{\text{wtot}} \{\text{pre}\} \ P \ \{\text{post}\}$

- $\vdash_{\text{tot}} \{\text{pre}\} \ P \ \{\text{post}\}$ implies $\models_{\text{tot}} \{\text{pre}\} \ P \ \{\text{post}\}$

Completeness for termination:

- every iterating set of rule schemata that terminates can always be proven to do so using $[!]_{\text{tot}}$

- i.e. there always exists a termination function for which the rule schemata are decreasing (under the invariant)
Conclusions and future work

Conclusions:

- recapped on GP and our verification research programme
- justified and defined two notions of total correctness
- added and demonstrated new proof rules for total correctness
- discussed our soundness and completeness results

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
Thank you! Danke schön!

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