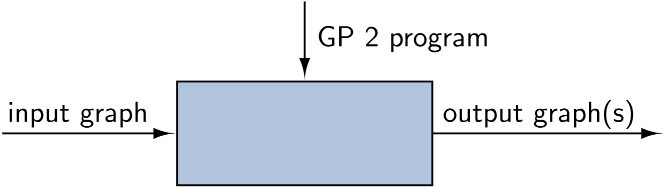
Hoare-Style Verification for GP 2

Detlef Plump [1] Chris Poskitt [2]

[1]The University of York

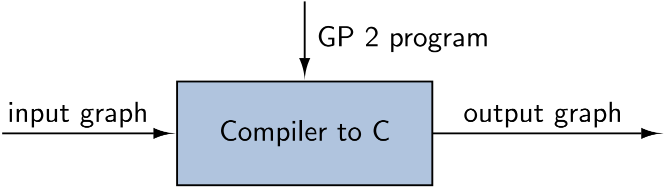
[2]ETH Zürich

Graph Programming Language GP 2



- Experimental DSL for graphs and graph-like structures
- Rule-based, visual manipulation of graphs
- Non-deterministic
- Simple syntax and semantics to facilitate formal reasoning

Graph Programming Language GP 2



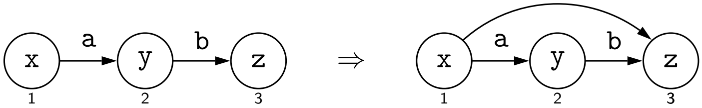
- Experimental DSL for graphs and graph-like structures
- Rule-based, visual manipulation of graphs
- Non-deterministic
- Simple syntax and semantics to facilitate formal reasoning

Program for transitive closure

A graph is *transitive* if for every directed path $v \rightsquigarrow v'$ with $v \neq v'$, there is an edge $v \to v'$

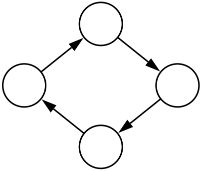
$\texttt{Main} = \texttt{link!}$

$\texttt{link}(\mathrm{a}, \mathrm{b}, \mathrm{x}, \mathrm{y}, \mathrm{z} \colon \texttt{list})$

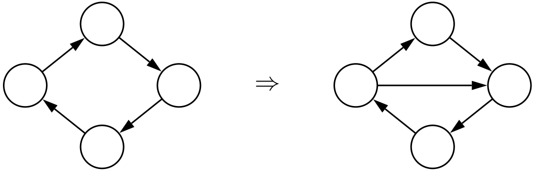


$\texttt{where not edge}(1, 3)$

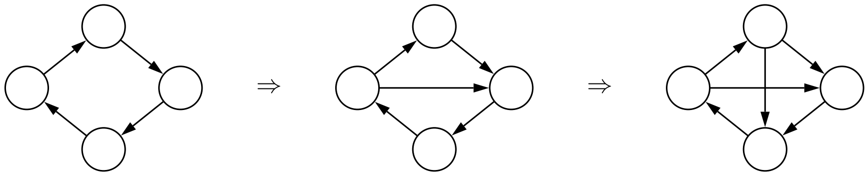
Program for transitive closure (cont'd)



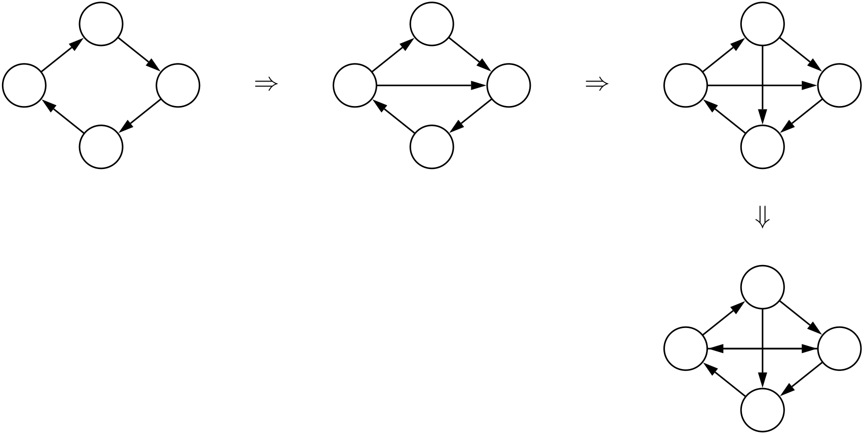
Program for transitive closure (cont'd)



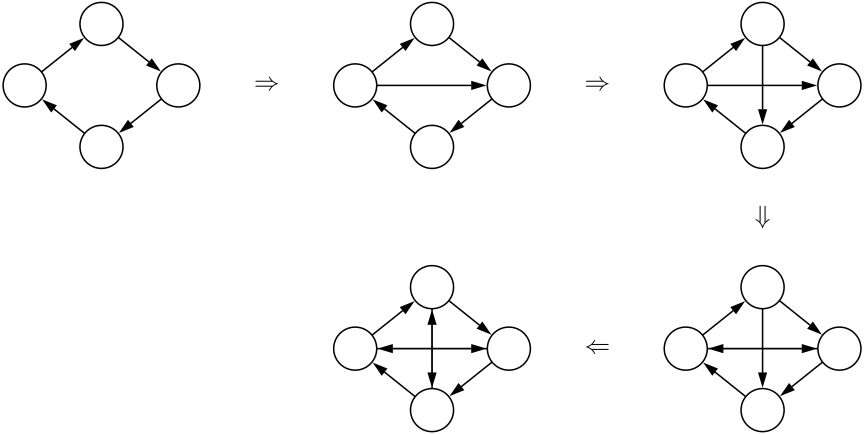
Program for transitive closure (cont'd)



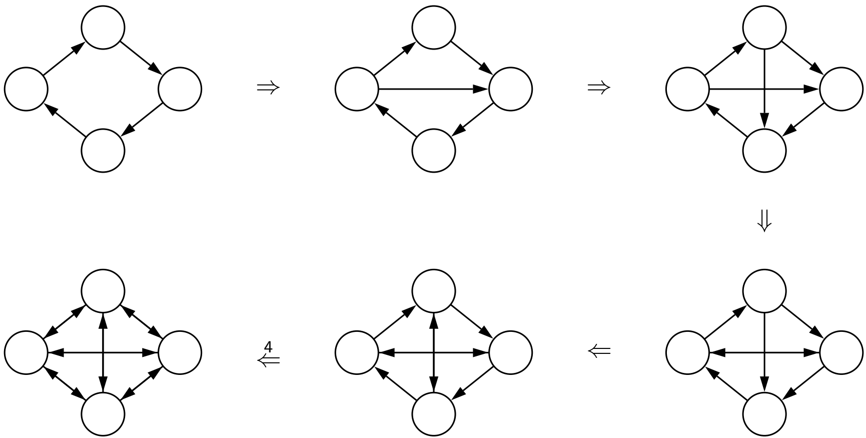
Program for transitive closure (cont'd)



Program for transitive closure (cont'd)

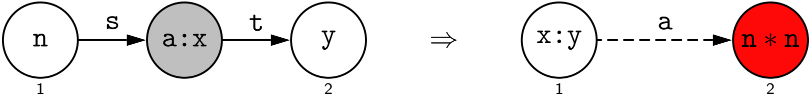


Program for transitive closure (cont'd)



Rule schemata ("attributed rules")

bridge(n: int; s, t: string; a: atom; x, y: list)



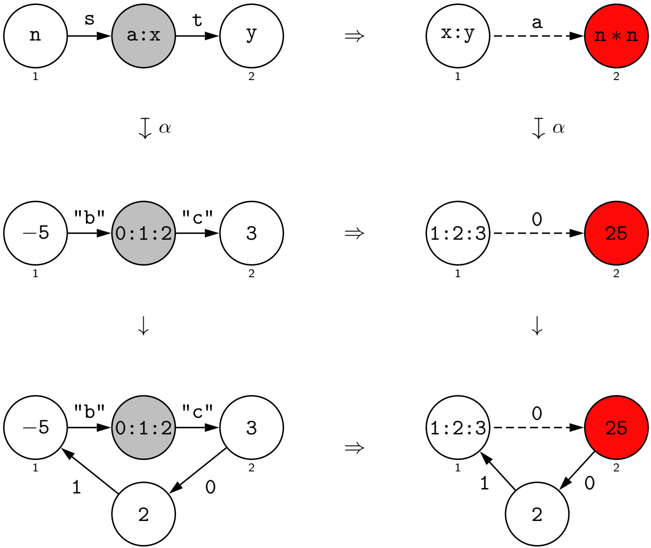
where n $<$ 0 and not edge(1, 3)

- ':' is list concatenation
- LHS expressions are *simple*
- Variables in RHS and condition occur in LHS
- Host graph expressions are constants (integers, strings and lists)
- *Marked* nodes and edges: red, green, blue, shaded, dashed

Subtype hierarchy for labels

$$\begin{tikzcd}
& \texttt{list} \arrow[d, phantom, sloped, "\supseteq"] & \\
& \texttt{atom} \arrow[dl, phantom, sloped, "\supseteq"] \arrow[dr, phantom, sloped, "\supseteq"] & \\
\texttt{int} & & \texttt{string} \arrow[d, phantom, sloped, "\supseteq"] \\
& & \texttt{char}
\end{tikzcd}$$

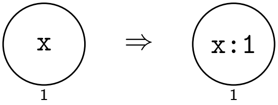
Rule-schema application



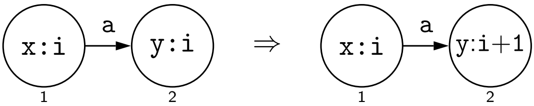
Program for vertex colouring

Main = init!; inc!

init(x: atom)



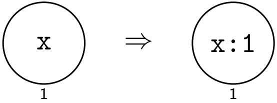
inc(x, y: atom; i: int; a: list)



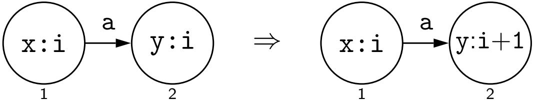
Program for vertex colouring

Main = init!; inc!

init(x: atom)

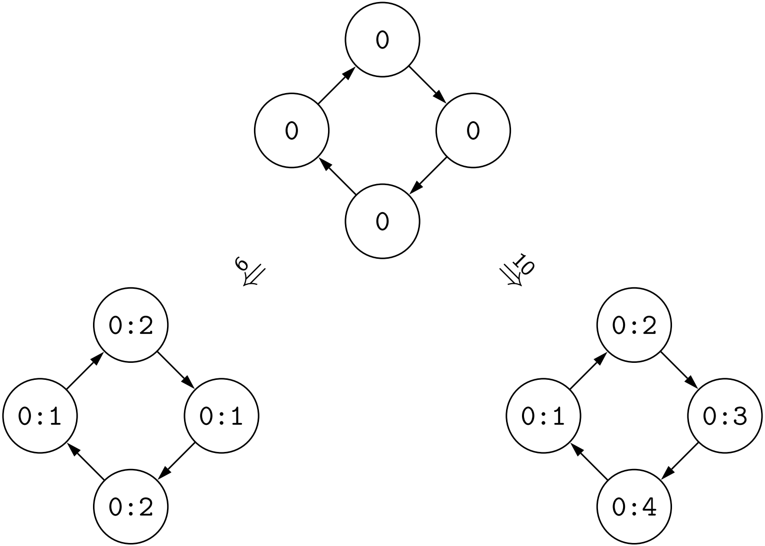


inc(x, y: atom; i: int; a: list)



Assumption: node labels in host graph are atoms

Program for vertex colouring (cont'd)



Programs: abstract syntax

$$
\begin{array}{lcl}
\text{Program} & ::= & \text{Decl } \{\text{Decl}\} \\
\vdots & & \\
\text{MainDecl} & ::= & \texttt{Main } `=' \text{ ComSeq} \\
\text{ComSeq} & ::= & \text{Com } \{`;' \text{ Com}\} \\
\text{Com} & ::= & \text{RuleId} \\
 & & \mid `\{' \ [\text{RuleId } \{`,' \text{ RuleId}\}] \ `\}' \\
 & & \mid \texttt{if} \text{ ComSeq } \texttt{then} \text{ ComSeq } [\texttt{else} \text{ ComSeq}] \\
 & & \mid \texttt{try} \text{ ComSeq } [\texttt{then} \text{ ComSeq } [\texttt{else} \text{ ComSeq}]] \\
 & & \mid \text{ComSeq } `!' \\
 & & \mid \ldots
\end{array}
$$

Transition relation defined by SOS

$\texttt{Main} = \{\texttt{r1}, \texttt{r2}\}; \{\texttt{r1}, \texttt{r2}\}; \texttt{r1!}$

$\texttt{r1} : ① \Rightarrow ① \qquad \texttt{r2} : ① \Rightarrow ②$

$$\begin{array}{ccccccc}
\langle \texttt{Main}, ① \rangle & \to & \langle P, ② \rangle & \to & \text{fail} & & \\
\downarrow & & & & & & \\
\langle P, ① \rangle & \to & \langle \texttt{r1!}, ① \rangle & \to & \langle \texttt{r1!}, ① \rangle & \to & \ldots \\
\downarrow & & & & & & \\
\langle \texttt{r1!}, ② \rangle & & & & & & \\
\downarrow & & & & & & \\
② & & & & & &
\end{array}$$

where $P = \{\texttt{r1}, \texttt{r2}\}; \texttt{r1!}$

Verification: Motivation

Motivated by applications of graph programs to defining semantics and analyses of languages and systems, e.g.

- Visual modelling/specification languages
- Model transformations
- Shape safety of pointer operations
- Concurrent asynchronous programming abstractions

Can existing "classical" verification tools help us?

Verification: Motivation

Motivated by applications of graph programs to defining semantics and analyses of languages and systems, e.g.

- Visual modelling/specification languages
- Model transformations
- Shape safety of pointer operations
- Concurrent asynchronous programming abstractions

Can existing "classical" verification tools help us?

- Yes, but blow up in inherently symmetric & dynamic problems

Verification: Motivation

Motivated by applications of graph programs to defining semantics and analyses of languages and systems, e.g.

- Visual modelling/specification languages
- Model transformations
- Shape safety of pointer operations
- Concurrent asynchronous programming abstractions

Can existing “classical” verification tools help us?

- Yes, but blow up in inherently symmetric & dynamic problems
- *Alternatively:* lift and tailor classical verification techniques to the domain of graphs and graph transformation

Verification: Our Approach

In particular: we are developing a theory to underpin assertional graph-based reasoning

- Graph/morphism-based assertion logic
- Hoare logics for programs
- Effective weakest precondition constructions for rules
- Exploits an algebraic characterisation of graph rewriting

$$\vdash \{pre\}\ P\ \{post\}$$

Graph-Based Assertion Logic

We combine ideas from:

- Nested conditions (finite)
  - Express local graph structure via morphisms
- Classical FO logic; attributed graph constraints
  - Express properties of attributes
- MSO logic on graphs (in the presentation of Courcelle)
  - Express non-local properties (paths, cycles, ...)

Graph-Based Assertion Logic

We combine ideas from:

- Nested conditions (finite)
  - Express local graph structure via morphisms
- Classical FO logic; attributed graph constraints
  - Express properties of attributes
- MSO logic on graphs (in the presentation of Courcelle)
  - Express non-local properties (paths, cycles, . . .)

Example: local graph structure and attributes

$$\forall_{\mathtt{L}} \mathtt{x}, \mathtt{y}.\ \forall\ \mathtt{x}_{\mathtt{v}}\ \ \mathtt{y}_{\mathtt{w}}\ [\,\mathtt{x}\ != \mathtt{y}\ \Rightarrow\ \exists\ \mathtt{x}_{\mathtt{v}} \xrightarrow{\mathtt{x}} \mathtt{y}_{\mathtt{w}}\,]$$

"for all pairs of nodes, if their labels are distinct, then they are adjacent"

Graph-Based Assertion Logic

Example: non-local property

$$\exists_V \mathtt{X}, \mathtt{Y}.\ \ \forall\ \bullet_{\mathtt{v}}\ [\ (\mathtt{v}\in\mathtt{X} \vee \mathtt{v}\in\mathtt{Y})\ \wedge \neg\,(\mathtt{v}\in\mathtt{X} \wedge \mathtt{v}\in\mathtt{Y})\ ]$$
$$\wedge\ \forall\ \bullet_{\mathtt{v}}\ \ \bullet_{\mathtt{w}}\ [\ \exists\ \bullet_{\mathtt{v}}\!\longrightarrow\!\bullet_{\mathtt{w}}\ \Rightarrow \neg\,(\mathtt{v}\in\mathtt{X} \wedge \mathtt{w}\in\mathtt{X})\ \wedge \neg\,(\mathtt{v}\in\mathtt{Y} \wedge \mathtt{w}\in\mathtt{Y})\ ]$$

"the graph is 2-colourable (bipartite)"

Graph-Based Assertion Logic

Example: non-local property

$$\exists_V X, Y.\ \forall \bullet_v\ [\,(v \in X \vee v \in Y) \wedge \neg(v \in X \wedge v \in Y)\,]$$
$$\wedge\ \forall \bullet_v\ \bullet_w\ [\,\exists\ \bullet_v \!\longrightarrow\! \bullet_w\ \Rightarrow \neg(v \in X \wedge w \in X) \wedge \neg(v \in Y \wedge w \in Y)\,]$$

"the graph is 2-colourable (bipartite)"

Example: attributes and non-local property

$$\exists\ \bullet_v\ [\neg\exists\ \bullet_v\ \bullet_w\ ] \wedge \forall_I x.\ \forall\ x_v\ [\,x > 0 \Rightarrow \exists\ x_v\ \bullet_w\ [\mathtt{path}(v, w)\,]\,]$$

"there is an (arbitrary-length) path from every positive integer-labelled node to a unique red node"

Hoare-Style Proof Calculi

Proof rules for (partial/total) correctness of GP 2 constructs, e.g.

$$[\text{comp}]\ \frac{\{c\}\ P\ \{e\} \quad \{e\}\ Q\ \{d\}}{\{c\}\ P; Q\ \{d\}} \qquad [!]\ \frac{\{inv\}\ \mathcal{R}\ \{inv\}}{\{inv\}\ \mathcal{R}!\ \{inv \wedge \neg \mathrm{App}(\mathcal{R})\}}$$

Core of the calculi:

$$\vdash \{\mathsf{Pre}(r, post)\}\ r\ \{post\}$$

Hoare-Style Proof Calculi

Proof rules for (partial/total) correctness of GP 2 constructs, e.g.

$$[\text{comp}]\ \frac{\{c\}\ P\ \{e\} \quad \{e\}\ Q\ \{d\}}{\{c\}\ P; Q\ \{d\}} \qquad [!]\ \frac{\{inv\}\ \mathcal{R}\ \{inv\}}{\{inv\}\ \mathcal{R}!\ \{inv \wedge \neg \mathrm{App}(\mathcal{R})\}}$$

Core of the calculi:

$$\vdash \{\mathrm{Pre}(r, post)\}\ r\ \{post\}$$

$$\forall$$

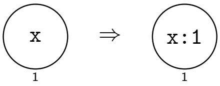
$$\{\ Pre(r, post)\ \}\ \ r{:}\, L \texttt{=>} R\ \ \{\ post\ \}$$

- Extended from Habel/Pennemann's construction to handle attributes, MSO logic, and path predicates

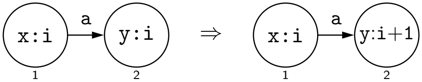
Example: Colouring

```
Main = init!; inc!
```

init(x: atom)



inc(x, y: atom; i: int; a: list)



$$c = \quad \forall_{L}\mathtt{a}.\ \forall\ (\mathtt{a})_{v}\ [\,\mathtt{atom(a)}\,]$$

$$d \wedge \neg\mathrm{App}(\{\mathtt{inc}\}) = \quad \forall_{L}\mathtt{a}.\ \forall\ (\mathtt{a})_{v}\ [\,\exists_{A}\mathtt{b}.\exists_{I}\mathtt{c}.\ \mathtt{a} = \mathtt{b:c} \wedge \mathtt{c} >= 1\,]$$

$$\wedge\ \neg\exists_{L}\mathtt{k}.\exists_{A}\mathtt{x},\mathtt{y}.\exists_{I}\mathtt{i}.\ \exists\ (\mathtt{x:i})_{v} \xrightarrow{\mathtt{k}} (\mathtt{y:i})_{w}$$

Example: Colouring

$$
\begin{prooftree}
\AxiomC{}
\LeftLabel{[ruleapp]}
\UnaryInfC{$\{\mathrm{Pre}(\mathtt{init}, e)\}\ \mathtt{init}\ \{e\}$}
\LeftLabel{[cons]}
\UnaryInfC{$\{e\}\ \mathtt{init}\ \{e\}$}
\LeftLabel{[!]}
\UnaryInfC{$\{e\}\ \mathtt{init!}\ \{e \wedge \neg\mathrm{App}(\{\mathtt{init}\})\}$}
\LeftLabel{[cons]}
\UnaryInfC{$\{c\}\ \mathtt{init!}\ \{d\}$}
\AxiomC{}
\LeftLabel{[ruleapp]}
\UnaryInfC{$\{\mathrm{Pre}(\mathtt{inc}, d)\}\ \mathtt{inc}\ \{d\}$}
\LeftLabel{[cons]}
\UnaryInfC{$\{d\}\ \mathtt{inc}\ \{d\}$}
\LeftLabel{[!]}
\UnaryInfC{$\{d\}\ \mathtt{inc!}\ \{d \wedge \neg\mathrm{App}(\{\mathtt{inc}\})\}$}
\LeftLabel{[comp]}
\BinaryInfC{$\vdash_{\mathrm{par}} \{c\}\ \mathtt{init!};\ \mathtt{inc!}\ \{d \wedge \neg\mathrm{App}(\{\mathtt{inc}\})\}$}
\end{prooftree}
$$

$$c = \quad \forall_{\mathtt{L}} \mathtt{a}.\ \forall\ (\mathtt{a})_{\mathtt{v}}\ [\,\mathtt{atom(a)}\,]$$

$$d \wedge \neg\mathrm{App}(\{\mathtt{inc}\}) = \quad \forall_{\mathtt{L}} \mathtt{a}.\ \forall\ (\mathtt{a})_{\mathtt{v}}\ [\,\exists_{\mathtt{A}}\mathtt{b}.\exists_{\mathtt{I}}\mathtt{c}.\ \mathtt{a} = \mathtt{b} : \mathtt{c} \wedge \mathtt{c} >= 1\,]$$
$$\wedge\ \neg\exists_{\mathtt{L}}\mathtt{k}.\exists_{\mathtt{A}}\mathtt{x}, \mathtt{y}.\exists_{\mathtt{I}}\mathtt{i}.\ \exists\ (\mathtt{x:i})_{\mathtt{v}} \xrightarrow{\mathtt{k}} (\mathtt{y:i})_{\mathtt{w}}$$

Example: Colouring

$$
\begin{prooftree}
\AxiomC{}
\LeftLabel{[ruleapp]}
\UnaryInfC{$\{\mathrm{Pre}(\mathtt{init}, e)\}\ \mathtt{init}\ \{e\}$}
\LeftLabel{[cons]}
\UnaryInfC{$\{e\}\ \mathtt{init}\ \{e\}$}
\LeftLabel{[!]}
\UnaryInfC{$\{e\}\ \mathtt{init!}\ \{e \wedge \neg \mathrm{App}(\{\mathtt{init}\})\}$}
\LeftLabel{[cons]}
\UnaryInfC{$\{c\}\ \mathtt{init!}\ \{d\}$}
\AxiomC{}
\LeftLabel{[ruleapp]}
\UnaryInfC{$\{\mathrm{Pre}(\mathtt{inc}, d)\}\ \mathtt{inc}\ \{d\}$}
\LeftLabel{[cons]}
\UnaryInfC{$\{d\}\ \mathtt{inc}\ \{d\}$}
\LeftLabel{[!]}
\UnaryInfC{$\{d\}\ \mathtt{inc!}\ \{d \wedge \neg \mathrm{App}(\{\mathtt{inc}\})\}$}
\LeftLabel{[comp]}
\BinaryInfC{$\vdash_{\mathrm{par}} \{c\}\ \mathtt{init!};\ \mathtt{inc!}\ \{d \wedge \neg \mathrm{App}(\{\mathtt{inc}\})\}$}
\end{prooftree}
$$

$$c = \quad \forall_{\mathtt{L}} \mathtt{a}.\ \forall\ (\mathtt{a})_{\mathtt{v}}\ [\mathtt{atom(a)}]$$

$$d \wedge \neg \mathrm{App}(\{\mathtt{inc}\}) = \quad \forall_{\mathtt{L}} \mathtt{a}.\ \forall\ (\mathtt{a})_{\mathtt{v}}\ [\exists_{\mathtt{A}} \mathtt{b}. \exists_{\mathtt{I}} \mathtt{c}.\ \mathtt{a} = \mathtt{b:c} \wedge \mathtt{c >= 1}]$$

$$\wedge\ \neg \exists_{\mathtt{L}} \mathtt{k}. \exists_{\mathtt{A}} \mathtt{x}, \mathtt{y}. \exists_{\mathtt{I}} \mathtt{i}.\ \exists\ (\mathtt{x:i})_{\mathtt{v}} \xrightarrow{\mathtt{k}} (\mathtt{y:i})_{\mathtt{w}}$$

$$\mathrm{Pre}(\mathtt{inc}, d) = \forall_{\mathtt{L}} \mathtt{k}. \forall_{\mathtt{A}} \mathtt{x}, \mathtt{y}. \forall_{\mathtt{I}} \mathtt{i}.\ \forall\ (\mathtt{x:i})_{\mathtt{v}} \xrightarrow{\mathtt{k}} (\mathtt{y:i})_{\mathtt{w}}$$

$$[\mathtt{i >= 1} \wedge \forall_{\mathtt{L}} \mathtt{a}.\ \forall\ (\mathtt{x:i})_{\mathtt{v}} \xrightarrow{\mathtt{k}} (\mathtt{y:i})_{\mathtt{w}}\ (\mathtt{a})$$

$$[\exists_{\mathtt{A}} \mathtt{b}. \exists_{\mathtt{I}} \mathtt{c}.\ \mathtt{a} = \mathtt{b:c} \wedge \mathtt{c >= 1}]\ ]$$

Open problems

Our calculi are sound with respect to the GP 2 semantics

But: assertion logic is not expressive

- Are there expressive extensions?
- Relative completeness (despite not expressive)?
- Proof rules for programs with composite tests / loop bodies?

Future Work

- Implementation for user-guided proofs
- Incorporating tracking information
- Automatic confluence and termination checking
- Applying to GROOVE's control programs; CEGAR