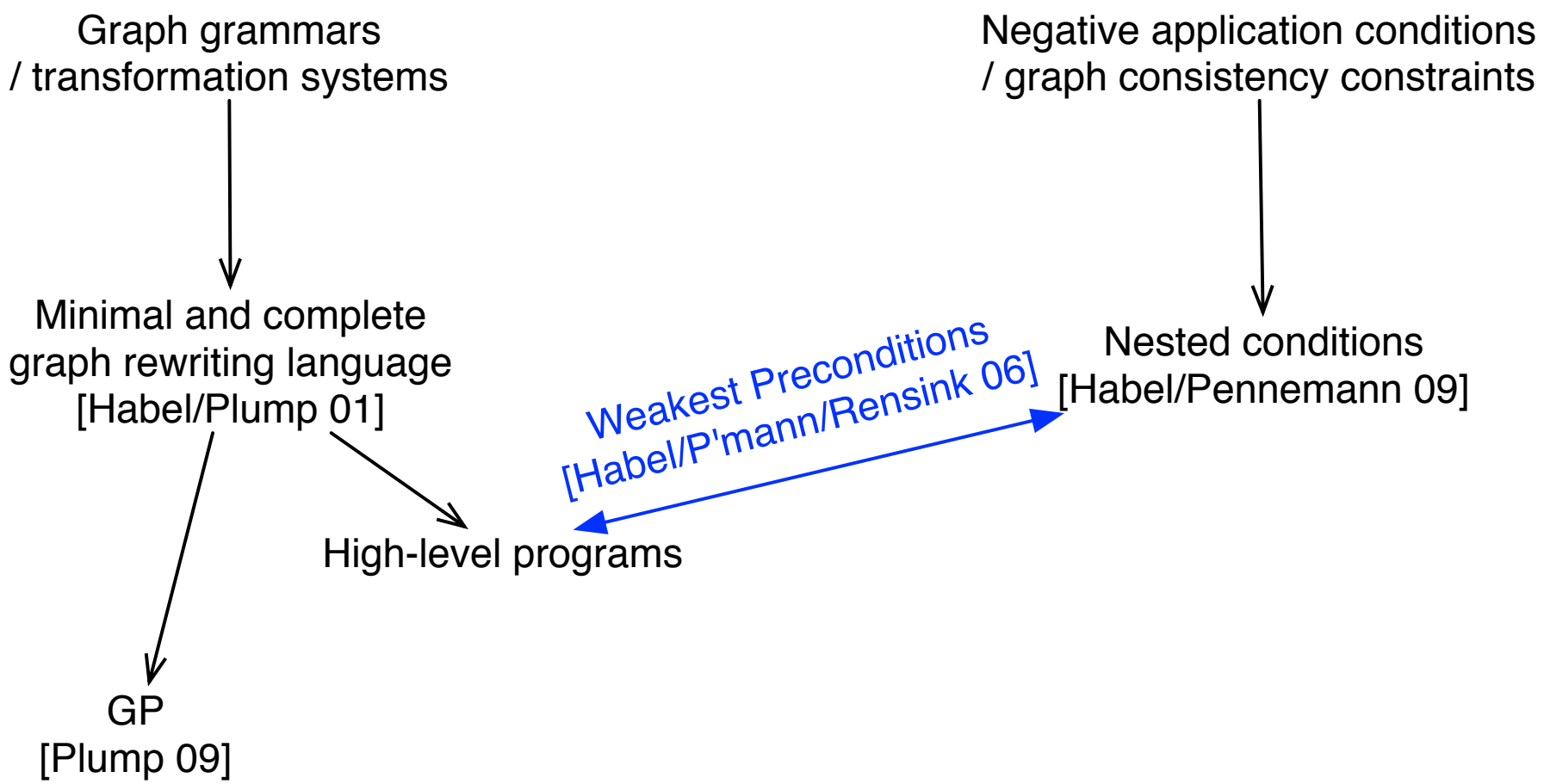
A Hoare Calculus for Graph Programs

Chris Poskitt and Detlef Plump

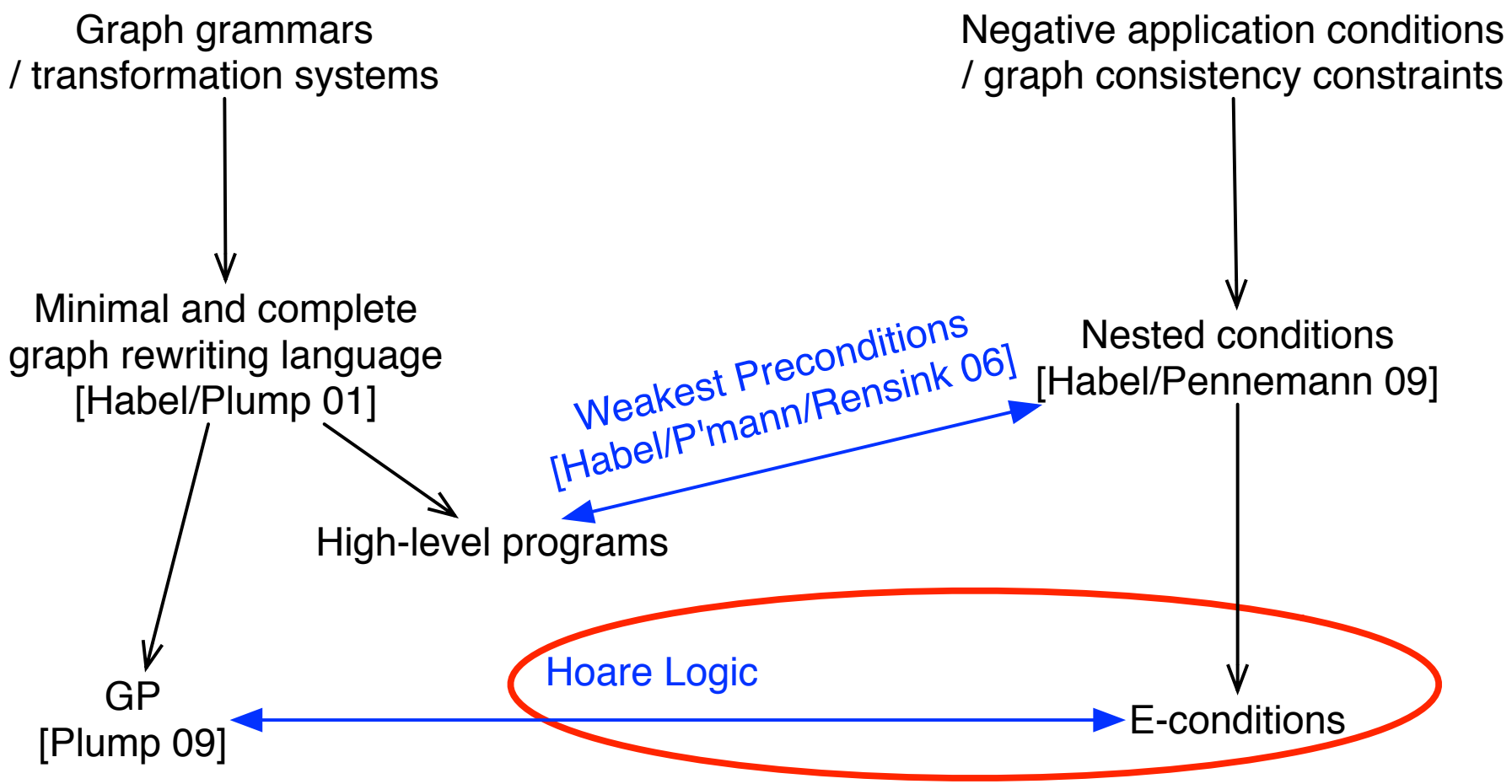
Department of Computer Science
The University of York, UK

ICGT 2010, 29th September 2010

Where this work fits into the picture



Where this work fits into the picture



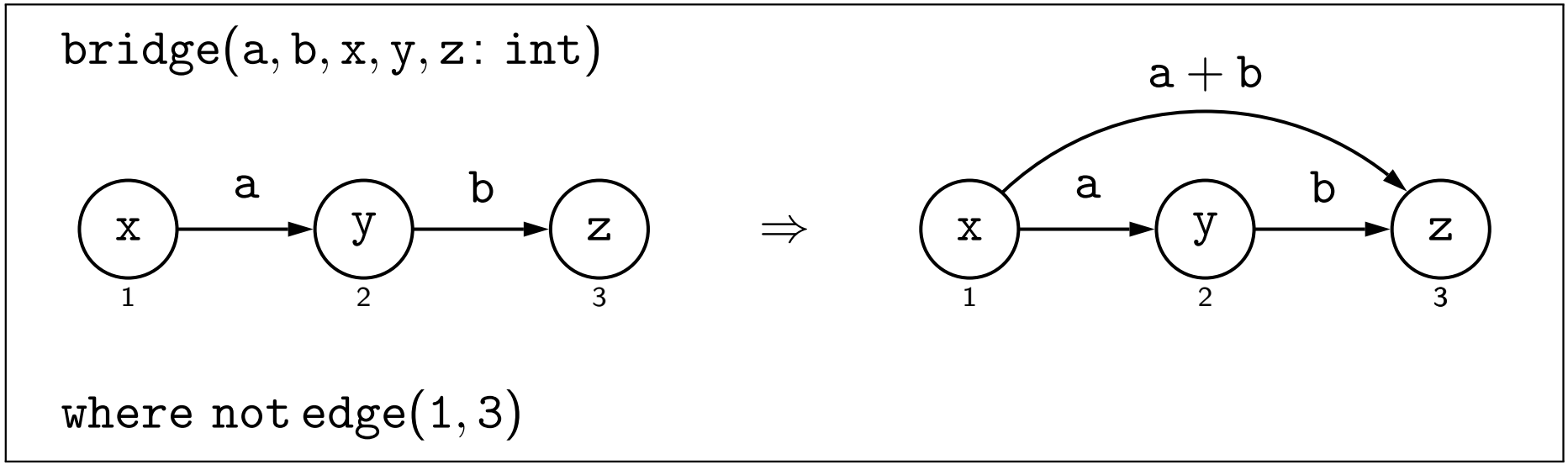
GP Overview

GP (for Graph Programs) is a nondeterministic, rule-based graph programming language.

It has a simple operational semantics, and aims to allow one to solve real problems with features like computation over graph labels.

Programs comprise a set of rule schemata, and a command sequence which directs their application to a provided input graph.

Rule Schemata

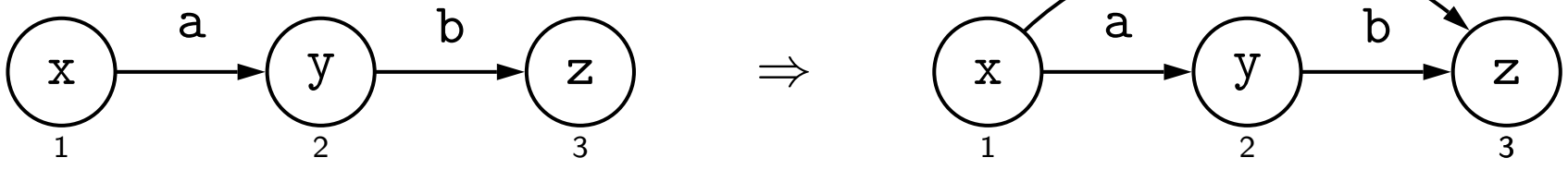


Rule schemata are syntactic constructs containing variables and expressions in labels, which are instantiated to (sequences of) integers and strings by graph matching.

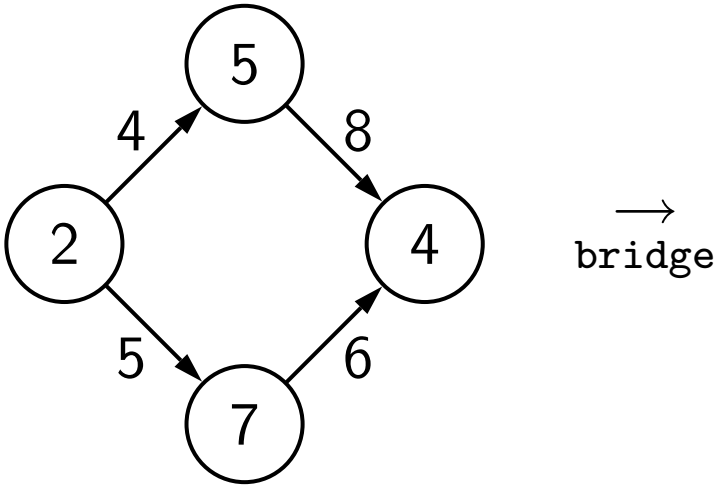
Rule schemata represent a potentially infinite set of "semantic" graph transformation rules.

Rule Schemata

`bridge(a, b, x, y, z: int)`



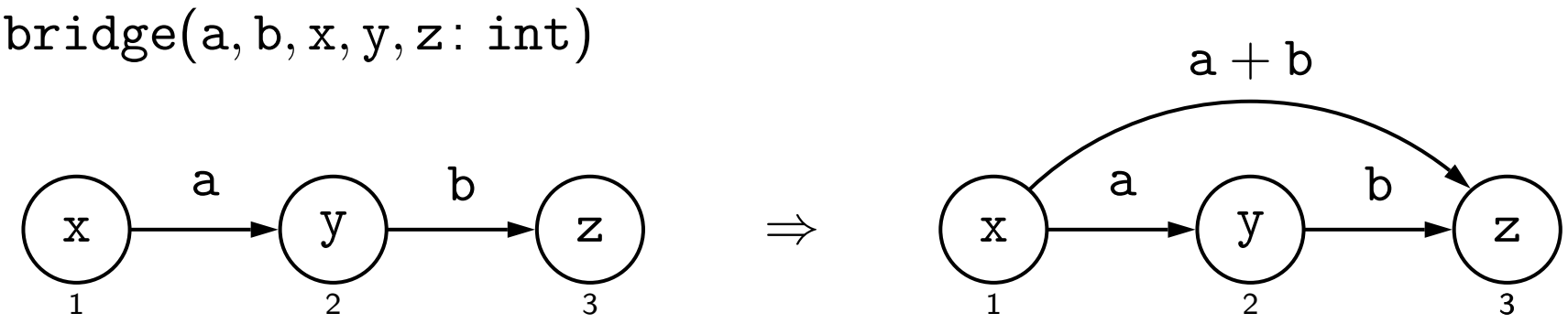
`where not edge(1, 3)`



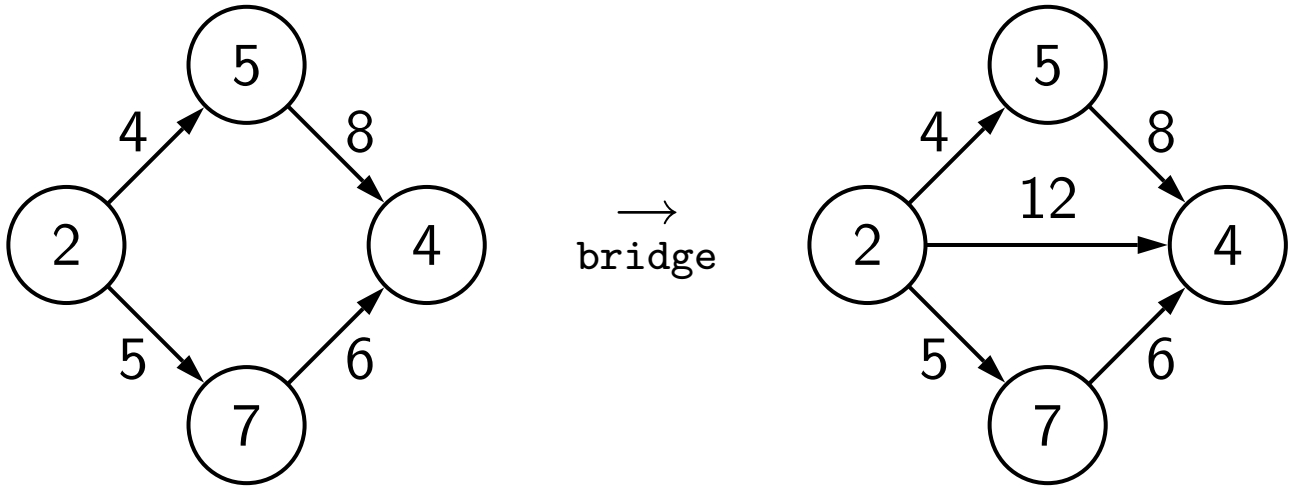
$\xrightarrow{\texttt{bridge}}$

Rule Schemata

bridge(a, b, x, y, z : int)

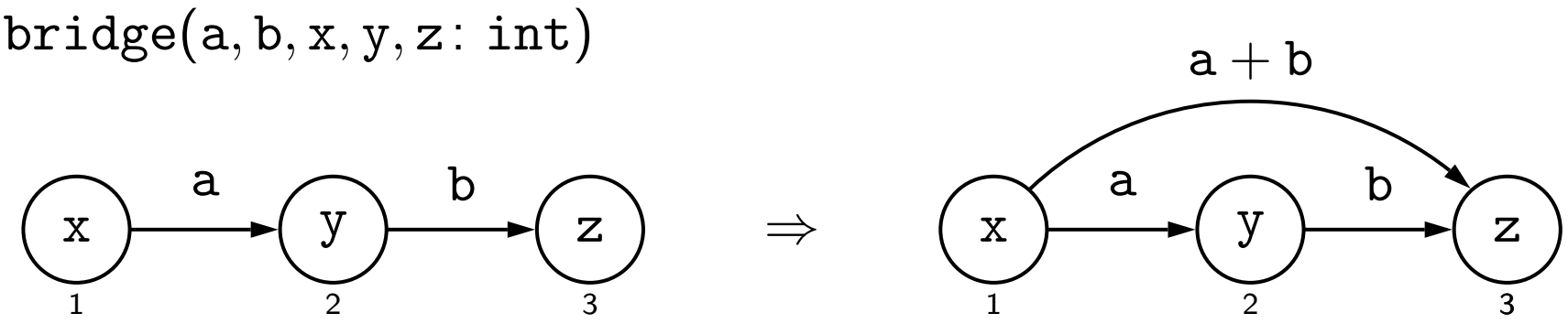


where not edge(1, 3)

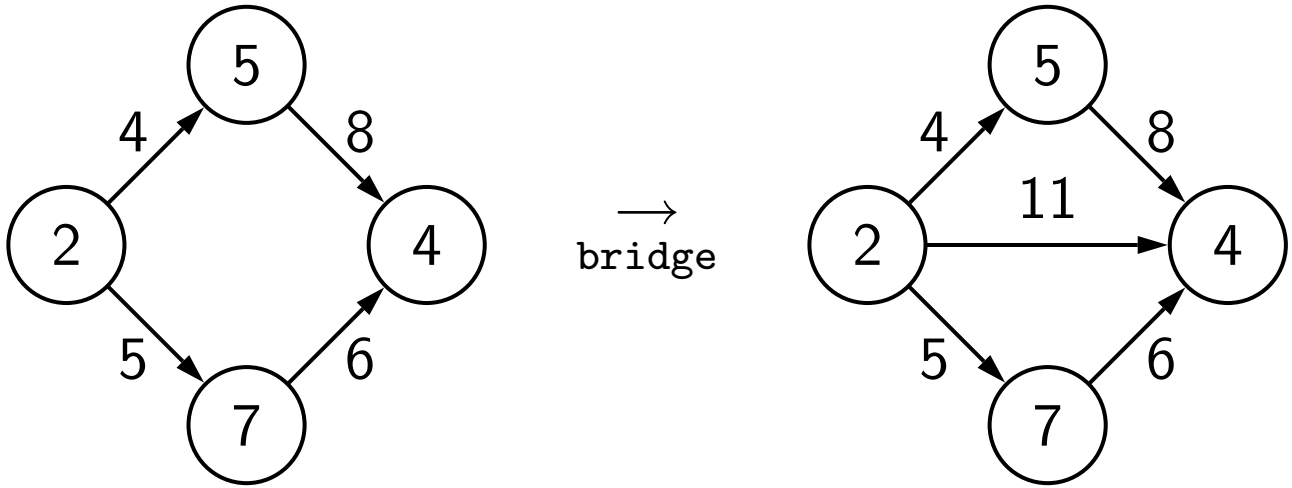


Rule Schemata

bridge(a, b, x, y, z : int)



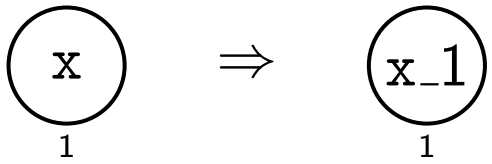
where not edge(1, 3)



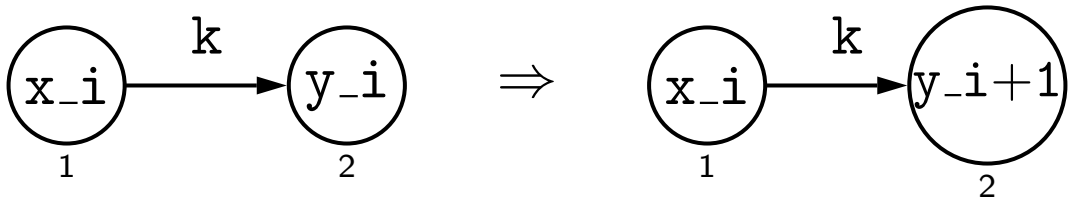
Graph Program: Running Example

```
main = init!; inc!
```

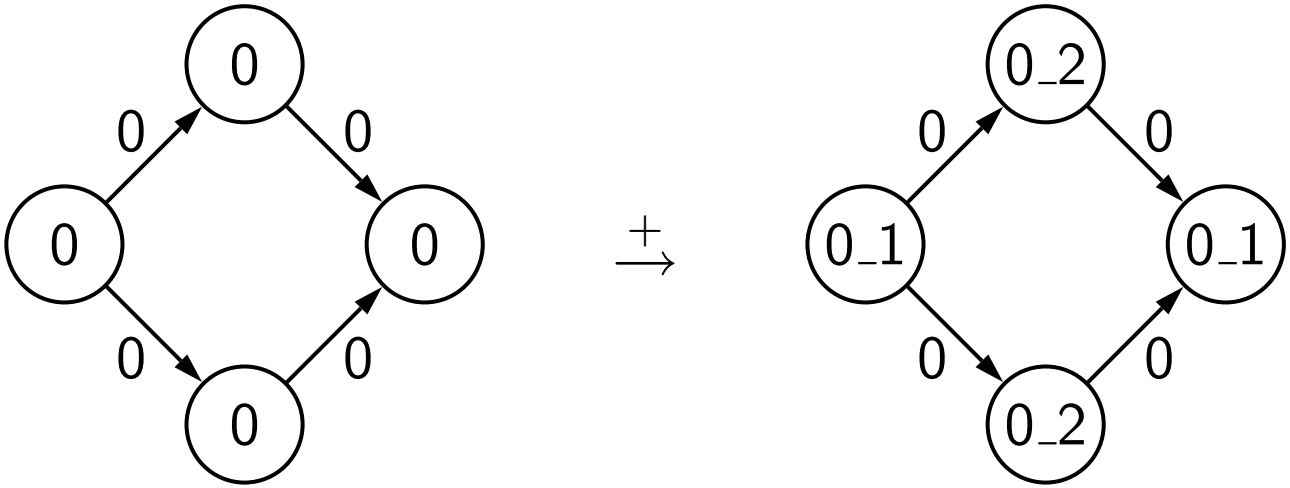
```
init(x: int)
```




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



Graph Program: Running Example



Describing the States of Graph Programs

To adopt Hoare logic to graph programs, we need a formalism for describing their states, i.e. directed graphs labelled over integers and strings.

Our formalism extends (nested) conditions [Habel/Pennemann 09], which are equivalently expressive to first-order formulae on graphs. Examples of conditions include:

$\exists(\textcircled{2} \to \textcircled{2})$

$\forall(\text{child}_1 , \exists(\text{child}_1 \to \text{parent}))$

Describing the States of Graph Programs

The infinite label alphabet of GP (sequences of strings and integers) means that many simple graph properties can only be expressed by infinite nested conditions.

Consider the property, "there is an integer labelled node". We require an infinite disjunction of conditions:

$$\exists(\ \textcircled{0}\ ) \vee \exists(\ \textcircled{1}\ ) \vee \exists(\ \textcircled{-1}\ ) \vee \exists(\ \textcircled{2}\ ) \vee \exists(\ \textcircled{-2}\ ) \vee \ldots$$

How can we write this finitely?

E-Conditions

E-conditions allow us to finitely express properties that require infinite nested conditions. e.g. "there is an integer labelled node",

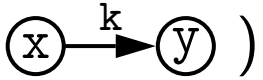
$$\exists(\ \textcircled{\mathtt{x}} \mid \mathrm{type}(\mathtt{x}) = \mathrm{int})$$

They are syntactic constructs, containing variables and expressions. These are instantiated and evaluated to integers and strings.

They represent a potentially infinite number of nested conditions.

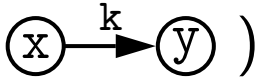
E-Conditions: Examples

"there exists a non-looping edge"

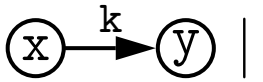
- $\exists($  $)$

E-Conditions: Examples

"there exists a non-looping edge"

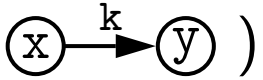
- $\exists($  $)$

"there exists a pair of adjacent integer-labelled nodes such that the label of the target is the square of that of the source"

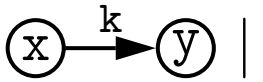
- $\exists($  $\mid \mathrm{type}(\mathtt{x},\mathtt{y}) = \mathrm{int} \wedge \mathtt{x} * \mathtt{x} = \mathtt{y})$

E-Conditions: Examples

"there exists a non-looping edge"

- $\exists($  $)$

"there exists a pair of adjacent integer-labelled nodes such that the label of the target is the square of that of the source"

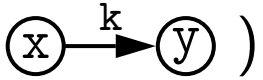
- $\exists($  $\mid \mathrm{type}(\mathtt{x},\mathtt{y}) = \mathrm{int} \wedge \mathtt{x} * \mathtt{x} = \mathtt{y})$

"no integer labelled node is incident to a non-looping outgoing edge"

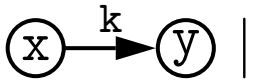
- $\forall($  $\mid \mathrm{type}(\mathtt{x}) = \mathrm{int}, \neg\exists($ $))$

E-Conditions: Examples

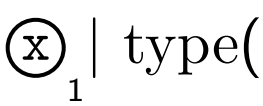
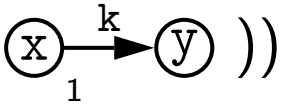
"there exists a non-looping edge"

- $\exists($  $)$

"there exists a pair of adjacent integer-labelled nodes such that the label of the target is the square of that of the source"

- $\exists($  $\mid \mathrm{type}(\mathtt{x},\mathtt{y}) = \mathrm{int} \wedge \mathtt{x} * \mathtt{x} = \mathtt{y})$

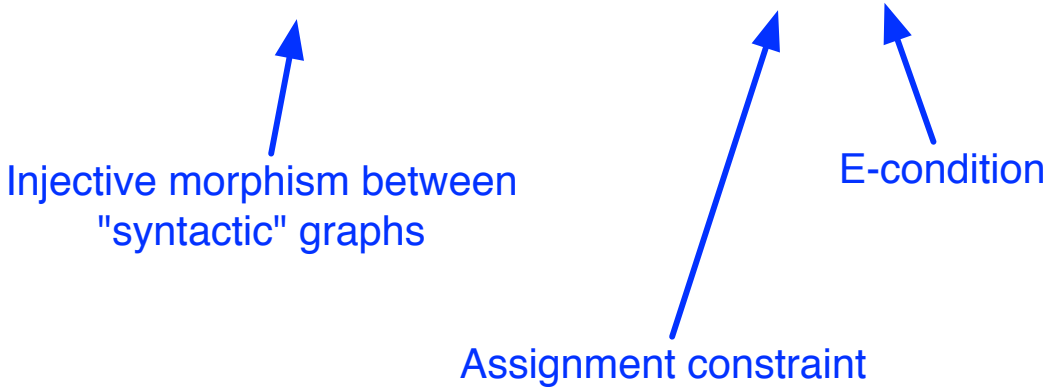
"no integer labelled node is incident to a non-looping outgoing edge"

- $\forall($  $\mid \mathrm{type}(\mathtt{x}) = \mathrm{int}, \neg\exists($  $))$

Note: can also write E-conditions over left- and right-hand sides of rule schemata, similar to (negative) application conditions.

E-conditions: Definition

$$\exists(a : P \hookrightarrow C | \gamma, c')$$



For "graph constraints", P is the empty graph, $\emptyset$.

For "(negative) application conditions", P is L or R of the rule schema.

The domain of the morphism in c' will be C.

E-conditions: Satisfaction

A graph G satisfies an E-condition $c = \exists(a\colon P \hookrightarrow C | \gamma, c')$, written $G \models c$, iff $\emptyset \hookrightarrow G \models c$.

E-conditions: Satisfaction

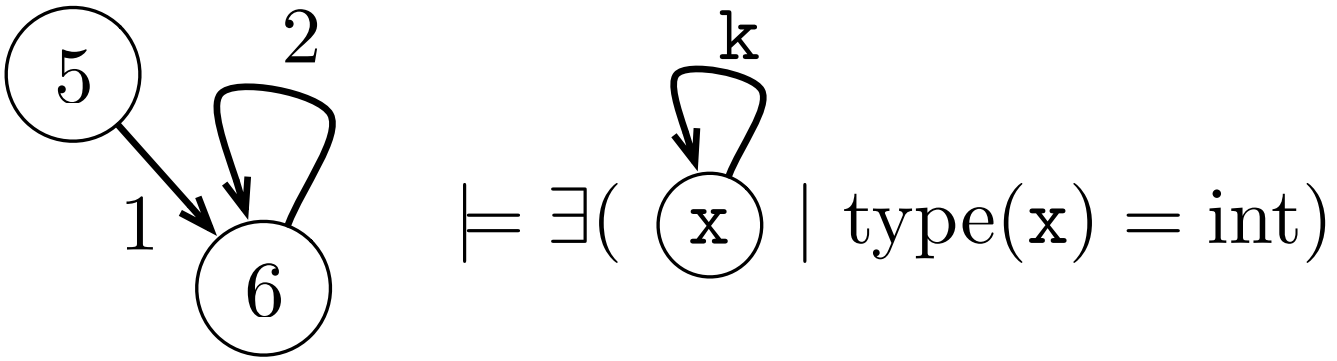
A graph G satisfies an E-condition $c = \exists(a\colon P \hookrightarrow C|\gamma, c')$, written $G \models c$, iff $\emptyset \hookrightarrow G \models c$.

A morphism $s\colon S \hookrightarrow G$ (of graphs over strings/integers) satisfies an E-condition if there is an assignment α from variables to (sequences of) character strings and integers s.t.

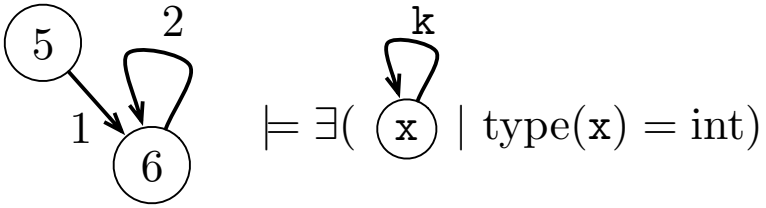
$$\begin{array}{ccc} S = P^\alpha & \hookrightarrow & C^\alpha \\ & \searrow^{=} \quad \swarrow_{q} & \\ & G & \end{array}$$

and $\gamma^\alpha = \mathtt{tt}$, and $q \models (c')^{\sigma_\alpha}$.

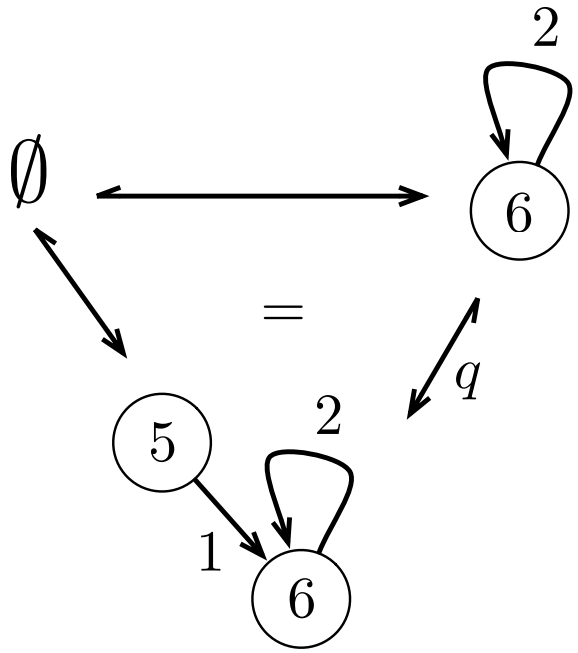
E-conditions: Satisfaction



E-conditions: Satisfaction



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



Also, $(\text{type}(\mathtt{x}) = \text{int})^{\alpha} = \mathtt{tt}$ and $q \models (c')^{\sigma_\alpha} = (\text{true})^{\sigma_\alpha} = \text{true}$.

Hoare Triples for Graph Programs

Given E-conditions c, d and a graph program P, a Hoare triple of the form $\{c\}\ P\ \{d\}$ can be read:

"whenever $G \models c$ and H results from running P on G, then $H \models d$"

Note: running P on G may yield

- different resulting graphs, or
- no resulting graph, because of non-termination or finite failure.

Axioms and Inference Rules

Sequential composition.

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

Rule of consequence.

$$[\text{cons}]\ c \Longrightarrow c'\ \frac{\{c'\}\ P\ \{d'\}}{\{c\}\ P\ \{d\}}\ d' \Longrightarrow d$$

Axioms and Inference Rules

If-then-else.

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

App is defined by a transformation that takes as input a set $\mathcal{R}$ of rule schemata, and transforms it into an E-condition describing the weakest property that a graph must satisfy for $\mathcal{R}$ to be applicable to it: LHS of some rule in $\mathcal{R}$ must occur in the current graph s.t. the dangling condition is satisfied.

Note: no Boolean guard.

Axioms and Inference Rules

As long as possible iteration (loop).

$$[!]\ \frac{\{inv\}\ \mathcal{R}\ \{inv\}}{\{inv\}\ \mathcal{R}!\ \{inv \wedge \neg \mathsf{App}(\mathcal{R})\}}$$

Axioms and Inference Rules

Set of rule schemata (none of which are applicable).

$$[\text{ruleset}_1]\ \frac{}{\{\neg \text{App}(\mathcal{R})\}\ \mathcal{R}\ \{\text{false}\}}$$

Set of rule schemata.

$$[\text{ruleset}_2]\ \frac{\{c\}\ r_1\ \{d\}\ \ldots\ \{c\}\ r_n\ \{d\}}{\{c\}\ \{r_1, \ldots, r_n\}\ \{d\}}$$

Axioms and Inference Rules

Rule schema application.

$$[\text{rule}]\ \frac{}{\{\text{Pre}(r, c)\}\ r\ \{c\}}$$

$\text{Pre}(r, c)$ is constructed from r and c such that if $G \models \text{Pre}(r, c)$ and the application of r to G results in a graph H, then $H \models c$.

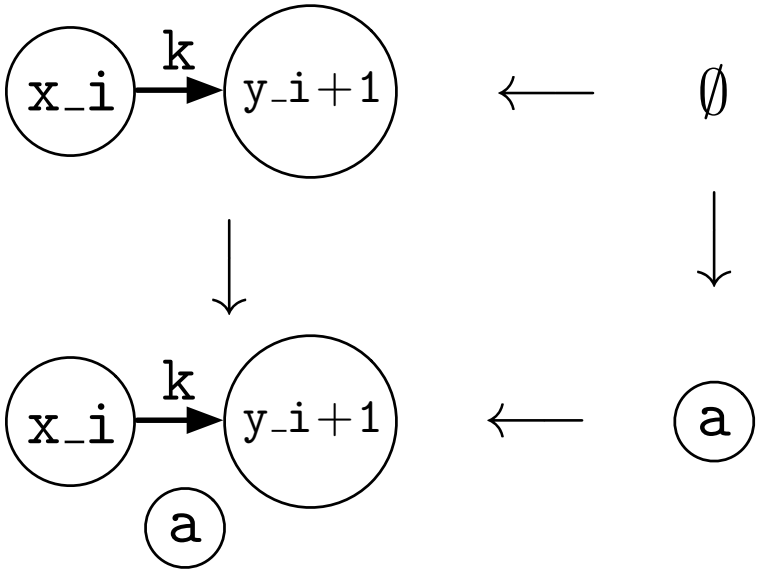
Transformation Pre (Example)

Consider r and c as follows:

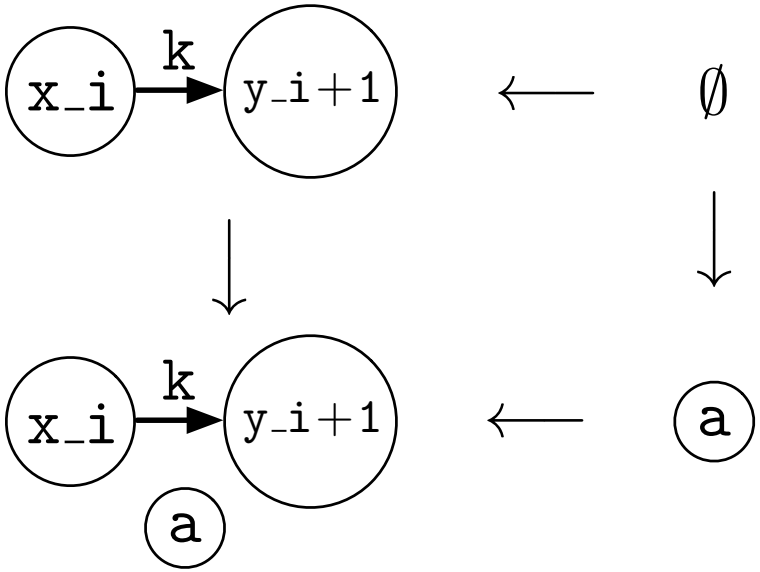
$$c = \neg\exists(\,\textcircled{\texttt{a}} \mid \mathrm{type}(\texttt{a}) = \mathrm{int})$$

$$r = \quad \underset{1}{\texttt{x\_i}} \xrightarrow{\texttt{k}} \underset{2}{\texttt{y\_i}} \quad \Rightarrow \quad \underset{1}{\texttt{x\_i}} \xrightarrow{\texttt{k}} \underset{2}{\texttt{y\_i+1}}$$

Transformation Pre (Example)



Transformation Pre (Example)

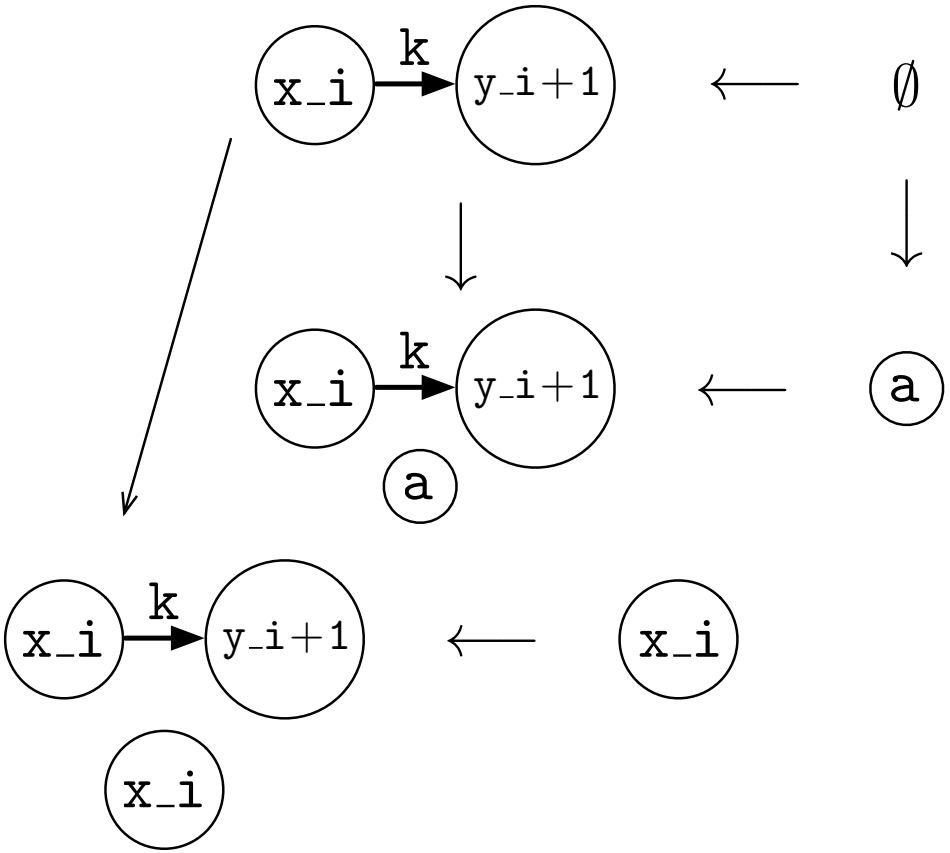


So we get a right E-app-condition:

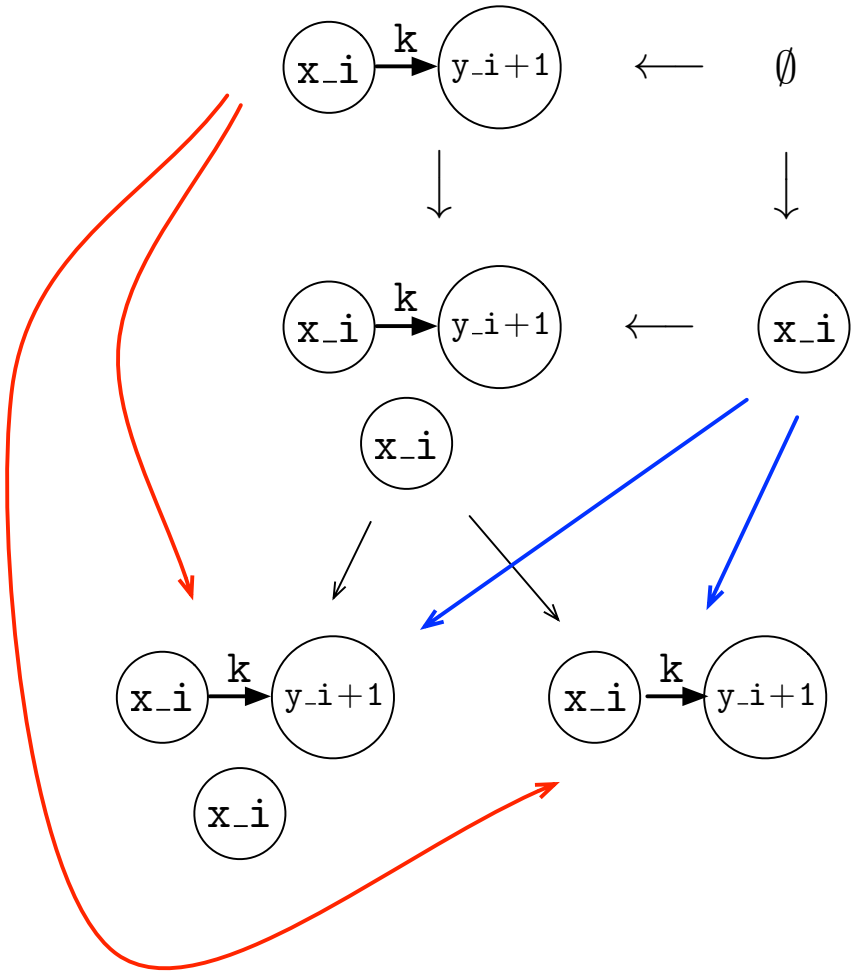
$$\neg\exists\Big(\texttt{x\_i} \xrightarrow{\texttt{k}} \texttt{y\_i+1} \hookrightarrow \begin{array}{c}\texttt{x\_i} \xrightarrow{\texttt{k}} \texttt{y\_i+1}\\ \texttt{a}\end{array} \mid \mathrm{type}(\texttt{a}) = \mathrm{int}\Big)$$

But what about overlappings of the condition and rule schema?

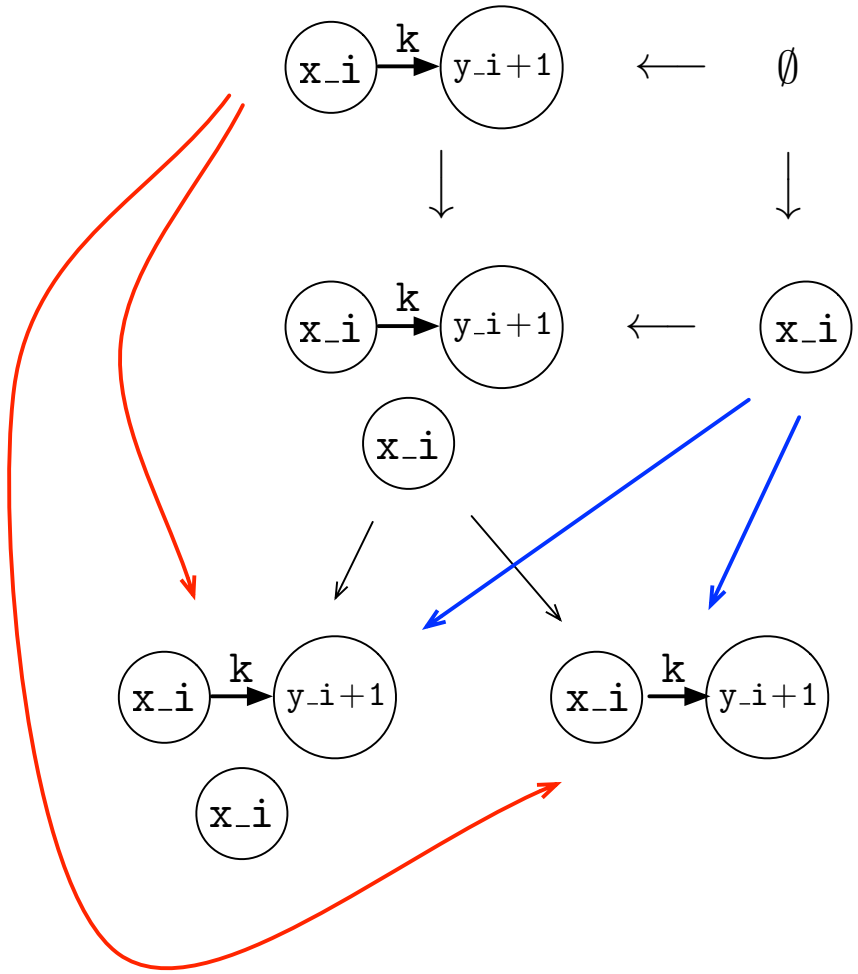
Transformation Pre (Example)



Transformation Pre (Example)



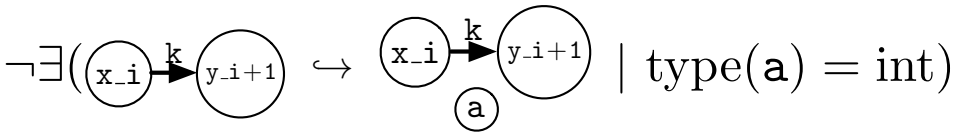
Transformation Pre (Example)



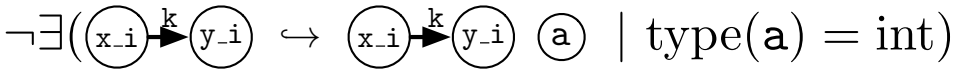
In this case, "overlappings" are irrelevant since `x_i` nor `y_i` are not single integers.

Transformation Pre (Example)

So we are now only dealing with:



Next, we shift this to be a left E-app-condition via a pushout construction.



Transformation Pre (Example)

Finally, we convert to a constraint (for all possible matches of L in a graph).

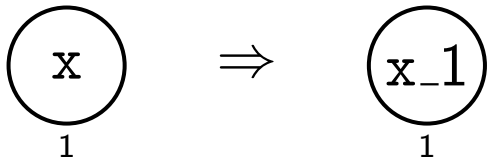
$$\forall(\emptyset \hookrightarrow \text{x\_i} \xrightarrow{\texttt{k}} \text{y\_i} \mid \text{type}(\texttt{x},\texttt{y},\texttt{k},\texttt{i}) = \text{int}),$$

$$\text{true} \Longrightarrow \neg\exists(\text{x\_i} \xrightarrow{\texttt{k}} \text{y\_i} \hookrightarrow \text{x\_i} \xrightarrow{\texttt{k}} \text{y\_i}\ \texttt{a} \mid \text{type}(\texttt{a}) = \text{int}))$$

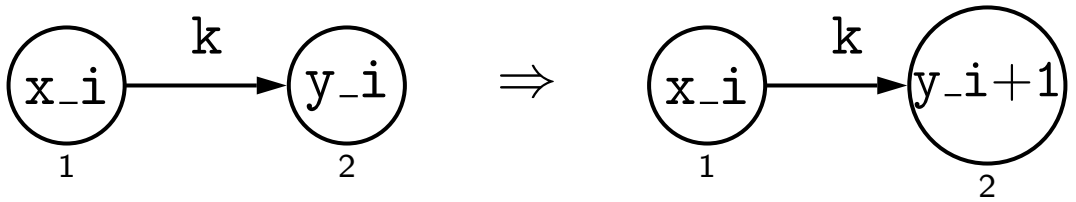
Graph Program: Running Example

```
main = init!; inc!
```

```
init(x: int)
```




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



Example Proof

We use the proof system to show the following:

$\{\neg\exists(\,\textcircled{a} \mid \mathrm{type}(a) \neq \mathrm{int})\}$

`init!; inc!`

$\{\forall(\,\textcircled{a}_1, \exists(\,\textcircled{a}_1 \mid a = b\_c \wedge \mathrm{type}(b, c) = \mathrm{int})) \wedge \neg\exists(\,\textcircled{x\_i} \xrightarrow{k} \textcircled{y\_i} \mid \mathrm{type}(i, k, x, y) = \mathrm{int})\}$

i.e. if the program is executed on a graph in which all node labels are integers, then any graph resulting will have the property that every node label is an integer with a colour attached, and that all adjacent nodes (by an integer-labelled edge) have distinct colours.

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

$$\begin{aligned}
c &= \neg\exists(\ \textcircled{a}\ \mid \mathsf{type(a)} \neq \mathsf{int}) \\
d &= \forall(\ \textcircled{a}\ , \exists(\ \textcircled{a}\ \mid \mathsf{a} = \mathsf{b_c} \wedge \mathsf{type(b, c)} = \mathsf{int})) \\
e &= \forall(\ \textcircled{a}^{1}_{1}\ , \exists(\ \textcircled{a}^{1}_{1}\ \mid \mathsf{type(a)} = \mathsf{int}) \vee \exists(\ \textcircled{a}_{1}\ \mid \mathsf{a} = \mathsf{b_c} \wedge \mathsf{type(b, c)} = \mathsf{int})) \\
\neg\mathrm{App}(\{\texttt{init}\}) &= \neg\exists(\ \textcircled{x}\ \mid \mathsf{type(x)} = \mathsf{int}) \\
\neg\mathrm{App}(\{\texttt{inc}\}) &= \neg\exists(\ \textcircled{x_i} \xrightarrow{k} \textcircled{y_i}\ \mid \mathsf{type(i, k, x, y)} = \mathsf{int}) \\
\mathrm{Pre}(\texttt{init}, e) &= \forall(\ \textcircled{x}_{1}\ \textcircled{a}_{2}\ \mid \mathsf{type(x)} = \mathsf{int}, \exists(\ \textcircled{x}_{1}\ \textcircled{a}_{2}\ \mid \mathsf{type(a)} = \mathsf{int}) \\
&\quad \vee \exists(\ \textcircled{x}_{1}\ \textcircled{a}_{2}\ \mid \mathsf{a} = \mathsf{b_c} \wedge \mathsf{type(b, c)} = \mathsf{int})) \\
\mathrm{Pre}(\texttt{inc}, d) &= \forall(\ \textcircled{x_i}^{k}_{1} \xrightarrow{} \textcircled{y_i}_{2}\ \textcircled{a}_{3}\ \mid \mathsf{type(i, k, x, y)} = \mathsf{int}, \\
&\quad \exists(\ \textcircled{x_i}^{k}_{1} \xrightarrow{} \textcircled{y_i}_{2}\ \textcircled{a}_{3}\ \mid \mathsf{a} = \mathsf{b_c} \wedge \mathsf{type(b, c)} = \mathsf{int}))
\end{aligned}$$

Soundness

Our main result is that the axioms and inference rules of our proof system are sound for proving partial correctness of graph programs.

That is, a graph program P is partially correct with respect to a precondition c and a postcondition d if there exists a full proof tree whose root is the triple $\{c\}\ P\ \{d\}$.

Conclusions

Concluding this talk, we have...

- Presented an overview of the first Hoare logic proof system for a practical graph transformation language.
- Introduced E-conditions as a graphical (but precise) formalism for finitely specifying many first-order graph properties.
- Defined axioms and inference rules that are sound in the sense of partial correctness w.r.t. GP's operational semantics.
- Demonstrated the proof system on a simple graph colouring program.

Future Work

Some future work for us:

- Prove completeness.
- Lift restrictions to [if] and [!].
- Total correctness.
- Make the proof system more expressive:
  - Does the output graph correspond correctly to the input graph?
  - Lift E-conditions to something more powerful, i.e. able to express MSOL (or non-local) graph properties — HR conditions [Habel/Radke 10]?
- Implementation in an interactive proof assistant, e.g. Coq or Isabelle.

The End

Many thanks for your attention!

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