

# Formalisms for the Verification of Graph Programs

Chris Poskitt

`cposkitt@cs.york.ac.uk`

Literature Review Seminar, 11th January 2010

# Today's Talk

1. Motivation for Reviewing Graph Formalisms
2. Nested Conditions and First Order Logic (FOL)
3. Monadic Second-Order Logic (MSOL)
4. Graph Reduction Specifications (GRSs)
5. Decidability
6. Conclusions and Questions

## A. Appendix

# Today's Talk

1. Motivation for Reviewing Graph Formalisms
2. Nested Conditions and First Order Logic (FOL)
3. Monadic Second-Order Logic (MSOL)
4. Graph Reduction Specifications (GRSs)
5. Decidability
6. Conclusions and Questions

## A. Appendix

## Motivation for Reviewing Graph Formalisms

- ▶ We can prove the correctness of imperative programs with respect to pre- and post-conditions [AO91, Rey98, NN07].
- ▶ e.g. Hoare logic allows us to prove the correctness of:

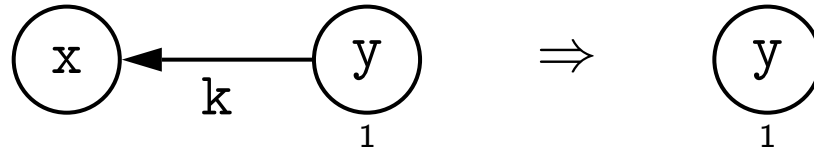
$$\{x = y\} x := x + 1; y := y + 1 \{x = y\}$$

- ▶ **Question:** can we prove the correctness of a *graph program* in a similar way?

## Motivation for Reviewing Graph Formalisms

- ▶ Consider the rule-based graph programming language GP [Plu09, Ste07].
- ▶ The *state* after each application of a rule is a graph [PS].

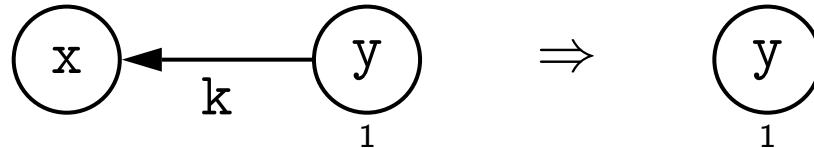
RemoveLeaf( $x, y, k : \text{int}$ ) =



## Motivation for Reviewing Graph Formalisms

- ▶ Consider the rule-based graph programming language GP [Plu09, Ste07].
- ▶ The *state* after each application of a rule is a graph [PS].

$\text{RemoveLeaf}(x, y, k : \text{int}) =$



- ▶ GP provides control constructs to control the application of rules to an input graph.
  - ▶ Sequential composition; **nondeterministic** application of a rule from a set; application for as long as possible; branching.

```
main = if (RemoveLeaf!; {TestNotIsolated, TestLoop}) then No else Yes.
```

## Motivation for Reviewing Graph Formalisms

- ▶ We might want to prove that the application of `RemoveLeaf` for as long as possible does not transform a tree into a non-tree.

$\{ \text{"graph is a tree"} \} \text{ RemoveLeaf! } \{ \text{"graph is a tree"} \}$

- ▶ Can we write *partial correctness proof axioms* for the application of graph transformation rules?
  - ▶ Need a notation to allow us to **precisely** describe and reason about graph properties.

# Today's Talk

1. Motivation for Reviewing Graph Formalisms
2. Nested Conditions and First Order Logic (FOL)
3. Monadic Second-Order Logic (MSOL)
4. Graph Reduction Specifications (GRSs)
5. Decidability
6. Conclusions and Questions

## A. Appendix

## Nested Conditions

- ▶ *Nested conditions* are a graphical and intuitive, yet precise formalism, for describing structural properties [HhP09, Pen09].
- ▶ They are equivalently expressive to first-order graph formulae [HhP09].
- ▶ Example:
  - ▶ “no node in graph  $G$  is isolated”
  - ▶ **In FOL:**

$$\forall x \in V_G \bullet \exists y \in V_G, a \in E_G \bullet x \neq y \wedge ((s(a) = x \wedge t(a) = y) \vee (s(a) = y \wedge t(a) = x))$$
  - ▶ **As a nested condition:**

$$\forall ( \bigcirc_1, \exists ( \bigcirc_1 \rightarrow \bigcirc ) \vee \exists ( \bigcirc \leftarrow \bigcirc_1 ) )$$

## Nested Conditions

- ▶ “every node is incident to **exactly one** loop”

- ▶  $\forall (O_1, \exists(\text{loop}_1) \wedge \neg \exists(\text{loop}_1))$

Not the full story!

## Nested Conditions

- ▶ “every node is incident to **exactly one** loop”

- ▶  $\forall ( \bigcirc_1, \exists ( \bigcirc_1 \text{ loop} ) \wedge \neg \exists ( \bigcirc_1 \text{ loop} ) )$

$$\equiv \forall ( \emptyset \rightarrow \bigcirc_1, \exists ( \bigcirc_1 \rightarrow \bigcirc_1 \text{ loop}, \text{true} ) \wedge \neg \exists ( \bigcirc_1 \rightarrow \bigcirc_1 \text{ loop}, \text{true} ) )$$

- ▶  $G \rightarrow H$  denotes an injective graph morphism from  $G$  to  $H$ .
  - ▶ i.e. a structure preserving mapping.
- ▶ These morphisms are not written when they can unambiguously be inferred [HhP09].

## Nested Conditions

- ▶ “every node is incident to **exactly one** loop”

$$\text{▶ } \forall ( \text{O}_1, \exists ( \text{O}_1 \text{ loop} ) \wedge \neg \exists ( \text{O}_1 \text{ loop} ) )$$

$$\equiv \forall ( \emptyset \rightarrow \text{O}_1, \exists ( \text{O}_1 \rightarrow \text{O}_1 \text{ loop}, \text{true} ) \wedge \neg \exists ( \text{O}_1 \rightarrow \text{O}_1 \text{ loop}, \text{true} ) )$$

$$\equiv \neg \exists ( \emptyset \rightarrow \text{O}_1, \neg \exists ( \text{O}_1 \rightarrow \text{O}_1 \text{ loop}, \text{true} ) \vee \exists ( \text{O}_1 \rightarrow \text{O}_1 \text{ loop}, \text{true} ) )$$

- ▶  $G \rightarrow H$  denotes an injective graph morphism from  $G$  to  $H$ .
  - ▶ i.e. a structure preserving mapping.
- ▶ These morphisms are not written when they can unambiguously be inferred [HhP09].

## Nested Conditions

- ▶ Our examples are in the category  $\langle \mathbf{Graphs}, Inj \rangle$ .
- ▶ Nested conditions can be generalised to describe properties of objects in any *weak adhesive high-level replacement (HLR) category*.
- ▶ The categories of graphs, hypergraphs, and Petri-nets are examples of weak adhesive HLR categories [EEPT06].

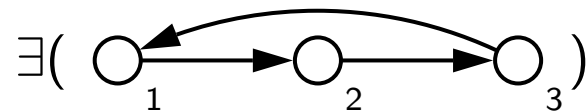
## Nested Conditions: What can be Expressed?

► Examples:

- “ $G$  is complete” [i.e.  $(|V_G| - 1)$ -regular]

$$\forall ( \bigcirc_1 \bigcirc_2, \exists ( \bigcirc_1 \rightarrow \bigcirc_2 ) )$$

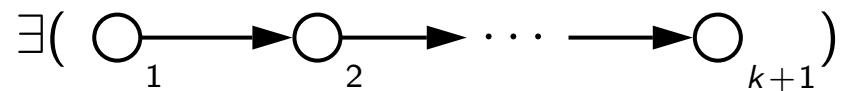
- “ $G$  contains a three-cycle”



- FOL / nested conditions can express  $k$ -regularity (fixed integer  $k$ ), node degree of at most  $k$ , that a graph is simple... [Cou90]

## Nested Conditions: What can be Expressed?

- ▶ FOL can only express *local properties of graphs* [Cou90].
  - ▶ i.e. the existence (or not) of fixed structures in the graph.
- ▶ What if we wanted to express “there exists a pair of nodes connected by a path”?
  - ▶ For a path of fixed length  $k \in \mathbb{N}$  we can write conditions of the following form for each value of  $k$ :



- ▶ But we cannot express the existence of a path of arbitrary length  $n \in \mathbb{N}$ .

## Nested Conditions: What can be Expressed?

- ▶ FOL is too weak for graphs!
- ▶ Very basic graph properties that FOL cannot express include [Cou]:
  - ▶ Is connected.
  - ▶ Is  $k$ -colourable (for fixed  $k \in \mathbb{N}$ ).
  - ▶ Is a tree.
  - ▶ Contains an arbitrary length cycle.
- ▶ More complex properties FOL cannot express include [Wil85, Har69]:
  - ▶ Is Hamiltonian — cannot express arbitrary length paths.
  - ▶ Is Eulerian — cannot express arbitrary length paths.
  - ▶ Is planar — cannot express arbitrary subdivisions of  $K_5$  or  $K_{3,3}$ .

## Finding Weakest Preconditions Using Nested Conditions

- ▶ To show the correctness of a program with respect to a pre- and post-condition, one can construct a *weakest precondition* and show that the precondition implies the weakest precondition.
  - ▶ A nested condition describing a postcondition of a graph program can be transformed into a weakest precondition for that program [HPR06, HP01].
  
- ▶ **Example:** find the weakest precondition for the application of  $\rho = \langle \text{loop} \Rightarrow \text{loopless} \rangle$  such that the resulting graph contains at least one loopless node.
  - ▶ **Weakest precondition:** the input graph contains either a loopless node, or a node incident to exactly one loop.

# Today's Talk

1. Motivation for Reviewing Graph Formalisms
2. Nested Conditions and First Order Logic (FOL)
3. Monadic Second-Order Logic (MSOL)
4. Graph Reduction Specifications (GRSs)
5. Decidability
6. Conclusions and Questions

## A. Appendix

## Monadic Second-Order Logic

- ▶ MSOL is an extension of FOL.
  - ▶ Introduces *set variables* and their quantification [Cou90].
- ▶ [Cou] denotes nodes by lowercase variables, sets of nodes by uppercase variables, and edges by a binary relation  $edg_G \subseteq V_G \times V_G$ .
- ▶ Set variables represent unary relations; the more powerful SOL allows variables denoting  $n$ -ary relations [Cou90].

## MSOL: Examples

**“graph is not empty and not connected”**

$$\exists X(\exists x(x \in X) \wedge \exists y(y \notin X) \wedge \forall x, y(\text{edg}(x, y) \Rightarrow (x \in X \Leftrightarrow y \in X)))$$

**“graph is 3-colourable”**

$$\exists X, Y, Z(\text{Part}(X, Y, Z) \wedge \forall x, y(\text{edg}(x, y) \wedge x \neq y \Rightarrow \neg(x \in X \wedge y \in X) \wedge \neg(x \in Y \wedge y \in Y) \wedge \neg(x \in Z \wedge y \in Z)))$$

## MSOL: Examples

**“graph is not empty and not connected”**

$$\exists X(\exists x(x \in X) \wedge \exists y(y \notin X) \wedge \forall x, y(\text{edg}(x, y) \Rightarrow (x \in X \Leftrightarrow y \in X)))$$

**“graph is 3-colourable”**

$$\exists X, Y, Z(\text{Part}(X, Y, Z) \wedge \forall x, y(\text{edg}(x, y) \wedge x \neq y \Rightarrow \neg(x \in X \wedge y \in X) \wedge \neg(x \in Y \wedge y \in Y) \wedge \neg(x \in Z \wedge y \in Z)))$$

$$\text{Part}(X, Y, Z) = \forall x((x \in X \vee x \in Y \vee x \in Z) \wedge (\neg(x \in X \wedge x \in Y) \wedge \neg(x \in Y \wedge x \in Z) \wedge \neg(x \in X \wedge x \in Z)))$$

## MSOL: What can be Expressed?

- ▶ MSOL can express many more graph properties than FOL [Cou90].
- ▶ Examples:
  - ▶ Is connected.
  - ▶ Is  $k$ -colourable (for fixed  $k \in \mathbb{N}$ ).
  - ▶ Is Hamiltonian (needs quantification over edges).
  - ▶ Is planar (using Kuratowski's [Wei] and Wagner's Theorems).

## MSOL: What can be Expressed?

- ▶ Cannot “count” in MSOL, hence properties we cannot express include [Cou90]:
  - ▶  $G$  has an even number of nodes
  - ▶ There are as many  $a$ 's as  $b$ 's, where  $a, b \in V_G \cup E_G$

# Today's Talk

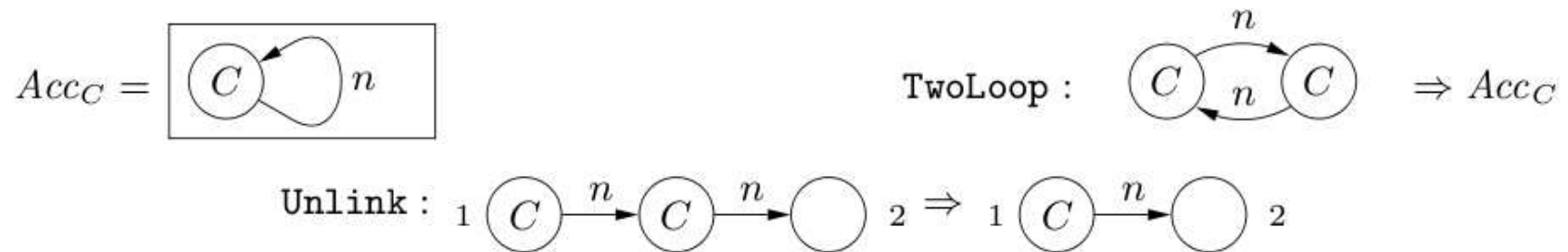
1. Motivation for Reviewing Graph Formalisms
2. Nested Conditions and First Order Logic (FOL)
3. Monadic Second-Order Logic (MSOL)
4. Graph Reduction Specifications (GRSs)
5. Decidability
6. Conclusions and Questions

## A. Appendix

## Graph Reduction Specifications

- ▶ Formalisms based on logics are not the only way in which graph properties can be described.
- ▶ *Graph Reduction Specifications* (GRSs) [BPR04, BPR03] are a powerful method for specifying shapes of graphs.
  - ▶ **Idea:** if a graph can be reduced to some *accepting graph* by the repeated application of a set of reduction rules, then it belongs to the class of shapes specified by that GRS.
- ▶ Can see a GRS as a graph grammar with the left- and right-hand sides of rules swapped, with the accepting graph corresponding to the starting graph of the grammar.

## GRSs: Specifying a Cycle Graph [BPR04]



- ▶ A cycle graph tested by this GRS remains a cycle graph at each stage of the reduction.
- ▶ The reduction rules do not create a cycle graph from an input graph that is not already one.

## GRSs: What can be Expressed?

- ▶ Very powerful: can specify every recursively enumerable set of graphs.
  - ▶ Follows from the result in [Ues78] that double pushout graph grammars have this property.
- ▶ GRSs can specify many graph properties that FOL cannot...
  - ▶ Is a tree
  - ▶ Is a balanced binary tree.
  - ▶ Is a cycle.
  - ▶ Is connected.
- ▶ ...and many that MSOL cannot:
  - ▶ Is a graph with an even number of nodes.

# Today's Talk

1. Motivation for Reviewing Graph Formalisms
2. Nested Conditions and First Order Logic (FOL)
3. Monadic Second-Order Logic (MSOL)
4. Graph Reduction Specifications (GRSs)
5. Decidability
6. Conclusions and Questions

## A. Appendix

## Decidability

### Validity Problem (VP)

**Instance:** a formula or GRS  $\gamma$ .

**Question (Formula):** does every graph  $G$  satisfy  $\gamma$  ( $\forall G. G \models \gamma$ )?

**Question (GRS):** is every graph  $G$  accepted by the GRS ( $\forall G. G \in L(\gamma)$ )?

### Satisfiability Problem (SP)

**Instance:** a formula or GRS  $\gamma$ .

**Question (Formula):** does there exist a graph  $G$  that satisfies  $\gamma$  ( $\exists G. G \models \gamma$ )?

**Question (GRS):** is  $L(\gamma) \neq \emptyset$ ?

### Model Checking Problem (MCP)

**Instance:** a formula or GRS  $\gamma$ , and a graph  $G$ .

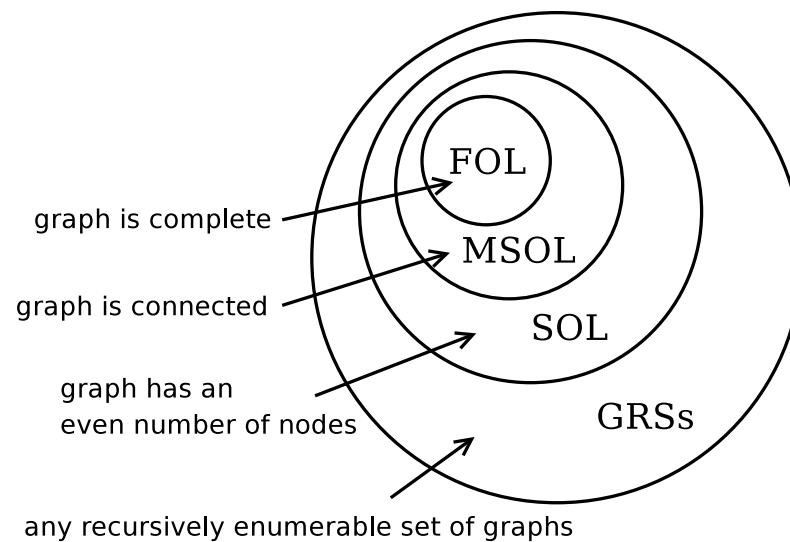
**Question (Formula):** does  $G$  satisfy  $\gamma$  ( $G \models \gamma$ )?

**Question (GRS):** is  $G \in L(\gamma)$  (the membership problem).

References: [Mar02, Hei02, BPR03]

## Decidability vs. Expressiveness

	FOL	MSOL	SOL	GRS	TGRS*
VP	No	No	No	No	?
SP	No	No	No	No	?
MCP	Yes	Yes	Yes	<b>No</b>	Yes



References: [Sto76]; \*TGRS = *Terminating* GRS.

# Today's Talk

1. Motivation for Reviewing Graph Formalisms
2. Nested Conditions and First Order Logic (FOL)
3. Monadic Second-Order Logic (MSOL)
4. Graph Reduction Specifications (GRSs)
5. Decidability
6. Conclusions and Questions

## A. Appendix

## Conclusions

- ▶ Studying graph formalisms for describing and reasoning about graph properties is an important first step towards verifying rule-based graph programs.
- ▶ Nested conditions are a graphical, intuitive, yet precise graph formalism equivalently expressive to FOL... and are therefore expressively weak for graphs.
- ▶ Extending such a formalism to the expressiveness of MSOL (or even SOL) would allow us to reason about many more graph properties.
- ▶ Expressibility can come at a cost to decidability.

# Thanks for your attention!

Slides and handout are available online:  
<http://www.cs.york.ac.uk/~cposkitt/>

[Extra material is in the appendix.]

# Today's Talk

1. Motivation for Reviewing Graph Formalisms
2. Nested Conditions and First Order Logic (FOL)
3. Monadic Second-Order Logic (MSOL)
4. Graph Reduction Specifications (GRSs)
5. Decidability
6. Conclusions and Questions

## A. Appendix

## Nested Condition Definition

- ▶ A (*nested*) condition  $c$  over a graph  $P$  is of the form  $c = \text{true}$  or  $c = \exists(a, c')$ .
  - ▶  $a : P \rightarrow C$  is a morphism;  $c'$  is a condition over  $C$ .

$$\begin{array}{ccc} \exists(P \xrightarrow{a} C, c') & & \\ p \searrow & & \swarrow q \\ & G & \end{array}$$

- ▶ A graph  $G$  *satisfies* a condition  $c$ , denoted  $G \models c$ , if the condition is over the empty graph and  $p$  satisfies  $c$ .
- ▶ A morphism  $p$  *satisfies* a condition  $c$ , denoted  $p \models c$ , if there exists a morphism  $q$  such that  $q \circ a = p$  and  $q \models c'$ .
  - ▶ Every graph and morphism satisfies true.
- ▶ Boolean formulas over conditions over  $P$  yield conditions over  $P$ , e.g.  $\neg c$  and  $\bigwedge_{j \in J} c_j$

## Expressiveness of First Order Logic

A property of a finite or infinite simple, directed graph is expressible by a closed first-order formula iff it is equivalent to a Boolean combination of properties of this form [Cou90]:

$$\exists v_1, \dots, v_s \left[ \bigwedge \{P(N(v_i, r)) \mid i \in [s]\} \wedge \bigwedge \{d(v_i, v_j) > 2r \mid 1 \leq i < j \leq s\} \right]$$

- ▶  $v_i$  denotes node  $i$ .
- ▶  $d(x, y)$  denotes the minimum length of a path between nodes  $x$  and  $y$ .
- ▶  $N(x, r)$  denotes the  $r$ -neighbourhood of  $x$ .
- ▶  $P$  is a first-order expressible property.



Krzysztof R. Apt and Ernst-Rüdiger Olderog.  
*Verification of Sequential and Concurrent Programs.*  
Springer-Verlag, 1991.



Adam Bakewell, Detlef Plump, and Colin Runciman.  
Specifying Pointer Structures by Graph Reduction, 2003.



Adam Bakewell, Detlef Plump, and Colin Runciman.  
Specifying Pointer Structures by Graph Reduction.  
volume 3062, pages 30–44. Springer-Verlag, 2004.



Bruno Courcelle.  
*Graph Algebras and Monadic Second-Order Logic.*  
Cambridge University Press, (in preparation).  
October 2009 draft accessed.



Bruno Courcelle.  
*Graph Rewriting: An Algebraic and Logic Approach*, volume B, book chapter/section 5.  
1990.



Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer.  
*Fundamentals of Algebraic Graph Transformation.*  
Springer, 2006.



Frank Harary.  
*Graph Theory.*  
Addison-Wesley, 1969.



James L. Hein.  
*Discrete Structures, Logic, and Computability.*  
Jones and Bartlett Publishers, Inc., 2nd edition, 2002.



Annegret Habel and Karl heinz Pennemann.

Correctness of high-level transformation systems relative to nested conditions.  
*Mathematical. Structures in Comp. Sci.*, 19:245–296, 2009.



Annegret Habel and Detlef Plump.

Computational completeness of programming languages based on graph transformation.  
*In Foundations of Software Science and Computation Structures : 4th International Conference, FOSSACS 2001: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings.*, pages 230+. 2001.



Annegret Habel, Karl Heinz Pennemann, and Arend Rensink.

Weakest Preconditions for High-Level Programs.  
volume 4178, pages 445–460. Springer-Verlag, 2006.



John Martin.

*Introduction to Languages and the Theory of Computation.*  
McGraw-Hill, 3rd edition, 2002.



Hanne Riis Nielson and Flemming Nielson.

*Semantics with Applications: An Appetizer.*  
Springer-Verlag, 2007.



Karl-Heinz Pennemann.

*Development of Correct Graph Transformation Systems.*  
PhD thesis, May 2009.



Detlef Plump.

The graph programming language GP.  
*In Proc. Algebraic Informatics (CAI 2009)*, volume 5725, pages 99–122. Springer-Verlag, 2009.



Detlef Plump and Sandra Steinert.

The semantics of graph programs.

In *Proc. Rule-Based Programming (RULE 2009)*, Electronic Proceedings in Theoretical Computer Science.  
To appear.



John C. Reynolds.

*Theories of Programming Languages*.  
Cambridge University Press, 1998.



Sandra Steinert.

*The Graph Programming Language GP*.  
PhD thesis, Department of Computer Science, The University of York, 2007.



L. Stockmeyer.

The polynomial-time hierarchy.  
*Theoretical Computer Science*, 3(1):1–22, October 1976.



Tadahiro Uesu.

A system of graph grammars which generates all recursively enumerable sets of labelled graphs.  
*Tsukuba Journal of Mathematics*, 2:11–26, 1978.



Eric W. Weisstein.

Kuratowski reduction theorem.  
From MathWorld—A Wolfram Web Resource.  
<http://mathworld.wolfram.com/KuratowskiReductionTheorem.html> Retrieved on 2010-01-08.



Robin J. Wilson.

*Introduction to Graph Theory*.  
Longman Scientific & Technical, 3rd edition, 1985.