Class Notes for CS 819 C - Spring 2002

# 2. Expressions, Commands, and Assertions

John C. Reynolds
Department of Computer Science
Carnegie Mellon University

revised May 1, 2002

Separation logic is an extension of Hoare logic [1, 2] to the specification and proof of programs in which data structures can contain several pointers to the same location, whose contents can be modified by the program.

The central idea, of an "separating conjunction" $p * q$ that asserts that $p$ and $q$ hold for separate parts of a data structure, is implicit in early work by Burstall [3]. It was explicitly described by Reynolds in lectures in the fall of 1999; then an intuitionistic logic based on this idea was discovered independently by Reynolds [19] and by Ishtiaq and O'Hearn [23]. Ishtiaq and O'Hearn realized that this logic was a special case of a bunched implication logic [18], and also introduced an "separating implication" operation.

In addition, Ishtiaq and O'Hearn devised a classical version of the logic that is more expressive than the intuitionistic version. (In particular, it can express storage deallocation.) Recently, Reynolds extended the classical version by adding pointer arithmetic [24, 25]; it is this extension that will be used in these lectures. We will also make use of O'Hearn's "frame rule" [30, 29], which permits local reasoning about components of programs, as well as recent work by O'Hearn on shared-variable concurrency.

Other recent relevant papers include work by Calcagno and O'Hearn on the treatment of garbage collection in the intuitionistic (and other) logics [22, 27, 28, 32], by Yang on a proof of the Schorr-Waite Algorithm [26, 29], and by Walker and Morrisett on a type system with goals that are similar to our logics [21].

Some of the research of my own that is described in these lectures was supported by National Science Foundation Grant CCR–9804014, and by BRICS at Aarhus University.

# References


[1] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580 and 583, October 1969.

[2] C. A. R. Hoare. Proof of a program: FIND. *Communications of the ACM*, 14(1):39–45, January 1971.

[3] Rodney M. Burstall. Some techniques for proving correctness of programs which alter data structures. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence 7*, pages 23–50. Edinburgh University Press, Edinburgh, Scotland, 1972.

[4] T. Kowaltowski. Correctness of programs manipulating data structures. Memorandum ERL-M404, University of California, Berkeley, California, September 1973.

[5] Stephen A. Cook and Derek C. Oppen. An assertion language for data structures. In *Conference Record of the Second ACM Symposium on Principles of Programming Languages*, pages 160–166, New York, 1975. ACM.

[6] Derek C. Oppen and Stephen A. Cook. Proving assertions about programs that manipulate data structures. In *Proceedings of Seventh Annual ACM Symposium on Theory of Computing*, pages 107–116, New York, 1975. ACM.

[7] Joseph M. Morris. A general axiom of assignment; assignment and linked data structures; a proof of the Schorr-Waite algorithm. In Manfried Broy and Gunther Schmidt,

editors, *Theoretical Foundations of Programming Methodology*, pages 25–51. D. Reidel, Dordrecht, Holland, 1982.

[8] Ian A. Mason. *The Semantics of Destructive Lisp*. CSLI Lecture Notes Number 5. Center for the Study of Language and Information, Menlo Park, CA, 1986.

[9] Ian A. Mason. Verification of programs that destructively manipulate data. *Science of Computer Programming*, 10(2):177–210, April 1988.

[10] Ian A. Mason and Carolyn Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1(3):287–327, July 1991.

[11] Furio Honsell, Ian A. Mason, Scott Smith, and Carolyn Talcott. A variable typed logic of effects. *Information and Computation*, 119(1):55–90, May 15, 1995.

[12] Ian A. Mason. A first order logic of effects. *Theoretical Computer Science*, 185(2):277–318, October 20, 1997.

[13] Ian D. B. Stark. *Names and Higher-Order Functions.* Ph. D. dissertation, University of Cambridge, Cambridge, England, December 1994.

[14] Ian D. B. Stark. Categorical models for local names. *Lisp and Symbolic Computation*, 9(1):77–107, February 1996.

[15] Ian D. B. Stark. Names, equations, relations: Practical ways to reason about *new*. *Fundamenta Informaticae*, 33:369–396, 1998.

[16] Andrew M. Pitts and Ian D. B. Stark. Operational reasoning for functions with local state. In Andrew D. Gordon and Andrew M. Pitts, editors, *Higher Order Operational Techniques in Semantics*, pages 227–273. Cambridge University Press, 1998.

[17] John C. Reynolds. *Theories of Programming Languages.* Cambridge University Press, Cambridge, England, 1998.

[18] Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 1999.

[19] John C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In Jim Davies, Bill Roscoe, and Jim Woodcock, editors, *Millennial Perspectives in Computer Science*, pages 303–321, Houndsmill, Hampshire, 2000. Palgrave.

[20] Cristiano Calcagno, Samin Ishtiaq, and Peter W. O'Hearn. Semantic analysis of pointer aliasing, allocation and disposal in Hoare logic. In Maurizio Gabbrielli and Frank Pfenning, editors, *Proceedings of the 2nd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'00)*, pages 190–201, New York, 2000. ACM Press.

[21] David Walker and Greg Morrisett. Alias types for recursive data structures. In Robert W. Harper, editor, *Types in Compilation*, volume 2071 of *Lecture Notes in Computer Science*, pages 177–206, Berlin, 2001. Springer-Verlag.

[22] Cristiano Calcagno. Program logics in the presence of garbage collection (abstract). In Fritz Henglein, John Hughes, Henning Makholm, and Henning Niss, editors, *SPACE 2001: Informal Proceedings of Workshop on Semantics, Program Analysis and Computing Environments for Memory Management*, page 11. IT University of Copenhagen, 2001.

[23] Samin Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 14–26, New York, 2001. ACM.

[24] John C. Reynolds. Lectures on reasoning about shared mutable data structure. IFIP Working Group 2.3 School/Seminar on State-of-the-Art Program Design Using Logic, Tandil, Argentina, September 6-13, 2000.

[25] John C. Reynolds and Peter W. O'Hearn. Reasoning about shared mutable data structure (abstract of invited lecture). In Fritz Henglein, John Hughes, Henning Makholm, and Henning Niss, editors, *SPACE 2001: Informal Proceedings of Workshop on Semantics, Program Analysis and Computing Environments for Memory Management*, page 7. IT University of Copenhagen, 2001. The slides for this lecture are available at ftp://ftp.cs.cmu.edu/user/jcr/spacetalk.ps.gz.

[26] Hongseok Yang. An example of local reasoning in BI pointer logic: The Schorr-Waite graph marking algorithm. In Fritz

Henglein, John Hughes, Henning Makholm, and Henning Niss, editors, *SPACE 2001: Informal Proceedings of Workshop on Semantics, Program Analysis and Computing Environments for Memory Management*, pages 41–68. IT University of Copenhagen, 2001.

[27] Cristiano Calcagno and Peter W. O'Hearn. On garbage and program logic. In *Foundations of Software Science and Computation Structures*, volume 2030 of *Lecture Notes in Computer Science*, pages 137–151, Berlin, 2001. Springer-Verlag.

[28] Cristiano Calcagno, Peter W. O'Hearn, and Richard Bornat. Program logic and equivalence in the presence of garbage collection. Submitted July 2001, 2001.

[29] Hongseok Yang. *Local Reasoning for Stateful Programs*. Ph. D. dissertation, University of Illinois, Urbana-Champaign, Illinois, July 2001.

[30] Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In Laurent Fribourg, editor, *Computer Science Logic*, volume 2142 of *Lecture Notes in Computer Science*, pages 1–19, Berlin, 2001. Springer-Verlag.

[31] Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In Ramesh Hariharan, Madhavan Mukund, and V. Vinay, editors, *FST TCS 2001: Foundations of Software Technology and Theoretical Computer Science*, volume 2245 of *Lecture Notes in*

*Computer Science*, pages 108–119, Berlin, 2001. Springer-Verlag.

[32] Cristiano Calcagno. *Semantic and Logical Properties of Stateful Programmming*. Ph. D. dissertation, Università di Genova, Genova, Italy, 2002.

[33] Hongseok Yang and Peter W. O'Hearn. A semantic basis for local reasoning. In M. Nielsen and U. Engberg, editors, *Foundations of Software Science and Computation Structures*, volume 2303 of *Lecture Notes in Computer Science*, pages 402–416, Berlin, 2002. Springer-Verlag.

[34] C. A. R. Hoare. Towards a theory of parallel programming. In C. A. R. Hoare and R. H. Perrott, editors, *Operating Systems Techniques*, volume 9 of *A.P.I.C. Studies in Data Processing*, pages 61–71, London, 1972. Academic Press.

[35] Susan Speer Owicki and David Gries. Verifying properties of parallel programs: An axiomatic approach. *Communications of the ACM*, 19(5):279–285, May 1976.

[36] Mads Tofte and Jean-Pierre Talpin. Implementation of the typed call-by-value $\lambda$-calculus using a stack of regions. In *Conference Record of POPL '94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 188–201, New York, 1994. ACM Press.

## Abstract Syntax of the Programming Language

$$
\begin{aligned}
\langle\text{exp}\rangle ::= {} & 0 \mid 1 \mid 2 \mid \cdots \\
& \mid \langle\text{var}\rangle \mid -\langle\text{exp}\rangle \mid \langle\text{exp}\rangle + \langle\text{exp}\rangle \mid \langle\text{exp}\rangle - \langle\text{exp}\rangle \\
& \mid \langle\text{exp}\rangle \times \langle\text{exp}\rangle \mid \cdots
\end{aligned}
$$

$$
\begin{aligned}
\langle\text{boolexp}\rangle ::= {} & \textbf{true} \mid \textbf{false} \\
& \mid \langle\text{exp}\rangle = \langle\text{exp}\rangle \mid \langle\text{exp}\rangle \neq \langle\text{exp}\rangle \mid \cdots \\
& \mid \neg\,\langle\text{boolexp}\rangle \mid \langle\text{boolexp}\rangle \wedge \langle\text{boolexp}\rangle \mid \langle\text{boolexp}\rangle \vee \langle\text{boolexp}\rangle \\
& \mid \langle\text{boolexp}\rangle \Rightarrow \langle\text{boolexp}\rangle \mid \langle\text{boolexp}\rangle \Leftrightarrow \langle\text{boolexp}\rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle\text{comm}\rangle ::= {} & \langle\text{var}\rangle := \langle\text{exp}\rangle \mid \textbf{skip} \mid \langle\text{comm}\rangle \,;\, \langle\text{comm}\rangle \\
& \mid \textbf{if}\ \langle\text{boolexp}\rangle\ \textbf{then}\ \langle\text{comm}\rangle\ \textbf{else}\ \langle\text{comm}\rangle \\
& \mid \textbf{while}\ \langle\text{boolexp}\rangle\ \textbf{do}\ \langle\text{comm}\rangle \\
& \mid \textbf{newvar}\ \langle\text{var}\rangle\ \textbf{in}\ \langle\text{comm}\rangle \\
& \mid \textbf{newvar}\ \langle\text{var}\rangle := \langle\text{exp}\rangle\ \textbf{in}\ \langle\text{comm}\rangle \\
& \mid \langle\text{var}\rangle := \textbf{cons}(\langle\text{exp}\rangle, \ldots, \langle\text{exp}\rangle) \\
& \mid \langle\text{var}\rangle := [\langle\text{exp}\rangle] \\
& \mid [\langle\text{exp}\rangle] := \langle\text{exp}\rangle \\
& \mid \textbf{dispose}\,\langle\text{exp}\rangle
\end{aligned}
$$

## Stores and Heaps

$$\begin{aligned}
\text{Atoms} &\subseteq \mathbf{Z} \\
\text{Locations} &\subseteq \mathbf{Z} \\
\text{Atoms} \cap \text{Locations} &= \{\} \\
\mathbf{nil} &\in \text{Atoms}
\end{aligned}$$

$$\begin{aligned}
\text{States} &= \text{Stores} \times \text{Heaps} \\
\text{Stores} &= \langle \text{var} \rangle \to \mathbf{Z} \\
\text{Heaps} &= \bigcup_{A \overset{\text{fin}}{\subseteq} \text{Locations}} (A \to \mathbf{Z})
\end{aligned}$$

Eventually, stores will map

$$\begin{array}{rcl}
\text{ordinary variables} & \text{into} & \mathbf{Z} \\
\text{sequence variables} & \text{into} & \text{sequences} \\
\text{tree variables} & \text{into} & \text{trees} \\
\text{assertion variables} & \text{into} & \text{Heaps} \to \mathbf{B} \\
& \vdots &
\end{array}$$

## Semantic Equations

$$\begin{aligned}
[\![-]\!]_{\text{exp}} &\in \langle \text{exp} \rangle \to \text{Stores} \to \mathbf{Z} \\
[\![-]\!]_{\text{boolexp}} &\in \langle \text{boolexp} \rangle \to \text{Stores} \to \mathbf{B} \\
[\![-]\!]_{\text{assert}} &\in \langle \text{assert} \rangle \to \text{Stores} \to \text{Heaps} \to \mathbf{B}
\end{aligned}$$

# A More Precise View

$$
\begin{aligned}
\mathrm{Stores}_V &= V \to \mathbf{Z} \\
\mathrm{Heaps} &= \bigcup_{A \overset{\mathrm{fin}}{\subseteq} \mathrm{Locations}} (A \to \mathbf{Z}) \\
\mathrm{States}_V &= \mathrm{Stores}_V \times \mathrm{Heaps},
\end{aligned}
$$

where $V \overset{\mathrm{fin}}{\subseteq} \langle \mathrm{var} \rangle$. Then

$$
\begin{aligned}
[\![ e \in \langle \exp \rangle ]\!]_{\exp} &\in \Big(\bigcup_{V \overset{\mathrm{fin}}{\supseteq} \mathrm{FV}(e)} \mathrm{Stores}_V\Big) \to \mathbf{Z} \\
[\![ b \in \langle \mathrm{boolexp} \rangle ]\!]_{\mathrm{boolexp}} &\in \Big(\bigcup_{V \overset{\mathrm{fin}}{\supseteq} \mathrm{FV}(b)} \mathrm{Stores}_V\Big) \to \mathbf{B} \\
[\![ p \in \langle \mathrm{assert} \rangle ]\!]_{\mathrm{assert}} &\in \Big(\bigcup_{V \overset{\mathrm{fin}}{\supseteq} \mathrm{FV}(p)} \mathrm{Stores}_V\Big) \to \mathrm{Heaps} \to \mathbf{B}.
\end{aligned}
$$

# The Semantics of Expressions

$$\llbracket - \rrbracket_{\mathrm{exp}} \in \langle \mathrm{exp} \rangle \to \mathrm{Stores} \to \mathbf{Z}.$$

$$\begin{aligned}
\llbracket 0 \rrbracket_{\mathrm{exp}} s &= 0 \qquad \text{(and similarly for 1, 2, ...)},\\
\llbracket v \rrbracket_{\mathrm{exp}} s &= s v,\\
\llbracket -e \rrbracket_{\mathrm{exp}} s &= -\llbracket e \rrbracket_{\mathrm{exp}} s,\\
\llbracket e_0 + e_1 \rrbracket_{\mathrm{exp}} s &= \llbracket e_0 \rrbracket_{\mathrm{exp}} s + \llbracket e_1 \rrbracket_{\mathrm{exp}} s\\
&\qquad \text{(and similarly for } -, \times, \ldots).
\end{aligned}$$

$$\llbracket - \rrbracket_{\mathrm{boolexp}} \in \langle \mathrm{boolexp} \rangle \to \mathrm{Stores} \to \mathbf{B}.$$

$$\begin{aligned}
\llbracket \mathbf{true} \rrbracket_{\mathrm{boolexp}} s &= \mathbf{true},\\
\llbracket \mathbf{false} \rrbracket_{\mathrm{boolexp}} s &= \mathbf{false},\\
\llbracket e_0 = e_1 \rrbracket_{\mathrm{boolexp}} s &= (\llbracket e_0 \rrbracket_{\mathrm{exp}} s = \llbracket e_1 \rrbracket_{\mathrm{exp}} s)\\
&\qquad \text{(and similarly for } \neq, \ldots),\\
\llbracket \neg b \rrbracket_{\mathrm{boolexp}} s &= \neg \llbracket b \rrbracket_{\mathrm{boolexp}} s,\\
\llbracket b_0 \wedge b_1 \rrbracket_{\mathrm{boolexp}} s &= \llbracket b_0 \rrbracket_{\mathrm{boolexp}} s \wedge \llbracket b_1 \rrbracket_{\mathrm{boolexp}} s\\
&\qquad \text{(and similarly for } \vee, \Rightarrow, \Leftrightarrow).
\end{aligned}$$

For example:

$$\begin{aligned}
\llbracket \mathsf{x} + \mathsf{y} = \mathsf{x} \rrbracket_{\mathrm{boolexp}} s &= (\llbracket \mathsf{x} + \mathsf{y} \rrbracket_{\mathrm{exp}} s = \llbracket \mathsf{x} \rrbracket_{\mathrm{exp}} s)\\
&= (\llbracket \mathsf{x} \rrbracket_{\mathrm{exp}} s + \llbracket \mathsf{y} \rrbracket_{\mathrm{exp}} s = \llbracket \mathsf{x} \rrbracket_{\mathrm{exp}} s)\\
&= (s\,\mathsf{x} + s\,\mathsf{y} = s\,\mathsf{x})\\
&= (s\,\mathsf{y} = 0).
\end{aligned}$$

## Free Variables

For any phrase $p$,

$$\mathrm{FV}(p) = \text{“The variables occurring free in } p\text{”}.$$

There are no binding constructions in expressions or boolean expressions, so that for these phrases $\mathrm{FV}(e)$ is the set of all variables occurring in $e$.

## Substitution

For any phrase $p$ such that $\mathrm{FV}(p) \subseteq \{v_1, \ldots, v_n\}$, we write

$$p/v_1 \to e_1, \ldots, v_n \to e_n$$

to denote the phrase obtained from $p$ by simulataneously substituting each expression $e_i$ for the variable $v_i$, (When there are bound variables in $p$, they will be renamed to avoid capture.)

## The Substitution Law for Expressions

**Proposition 1** *Let $\delta$ abbreviate the substitution*

$$v_1 \to e_1, \ldots, v_n \to e_n,$$

*let $s$ be a store such that* $\mathrm{FV}(e_1) \cup \cdots \cup \mathrm{FV}(e_n) \subseteq \mathrm{dom}\, s$, *and let*

$$\widehat{s} = [\, v_1 \colon [\![ e_1 ]\!]_{\mathrm{exp}} s \mid \ldots \mid v_n \colon [\![ e_n ]\!]_{\mathrm{exp}} s \,].$$

*If $e$ is an expression (or boolean expression) such that* $\mathrm{FV}(e) \subseteq \{v_1, \ldots, v_n\}$, *then*

$$[\![ e/\delta ]\!]_{\mathrm{exp}} s = [\![ e ]\!]_{\mathrm{exp}} \widehat{s}.$$

## Structural Operational Semantics

An execution (of a command) is a sequence

$$\gamma_0 \leadsto \gamma_1 \leadsto \cdots$$

of *configurations* $\gamma_i$ related by the *transition relation* $\leadsto$.

A configuration is either:

- *nonterminal*: A command-state pair $\langle c, (s, h)\rangle$, where $\mathrm{FV}(c) \subseteq \mathrm{dom}\, s$.
- *terminal*: A state $(s, h)$ or **abort**.

For example:

$$\begin{gathered}\langle \mathsf{x} := \mathsf{x} + 1 \,;\, \mathsf{y} := \mathsf{y} + \mathsf{x} \,;\, \mathsf{x} := \mathsf{x} + 1, ([\, \mathsf{x}\colon 3 \mid \mathsf{y}\colon 7\,], [\,])\rangle \leadsto \\ \langle \mathsf{y} := \mathsf{y} + \mathsf{x} \,;\, \mathsf{x} := \mathsf{x} + 1, ([\, \mathsf{x}\colon 4 \mid \mathsf{y}\colon 7\,], [\,])\rangle \leadsto \\ \langle \mathsf{x} := \mathsf{x} + 1, ([\, \mathsf{x}\colon 4 \mid \mathsf{y}\colon 11\,], [\,])\rangle \leadsto \\ ([\, \mathsf{x}\colon 5 \mid \mathsf{y}\colon 11\,], [\,]).\end{gathered}$$

$$\begin{gathered}\langle \mathsf{x} := \mathbf{cons}(1, 2) \,;\, \mathsf{y} := [\mathsf{x}] \,;\, [\mathsf{x} + 1] := 3, ([\, \mathsf{x}\colon 3 \mid \mathsf{y}\colon 4\,], [\,])\rangle \leadsto \\ \langle \mathsf{y} := [\mathsf{x}] \,;\, [\mathsf{x} + 1] := 3, ([\, \mathsf{x}\colon 37 \mid \mathsf{y}\colon 4\,], [\, 37\colon 1 \mid 38\colon 2\,])\rangle \leadsto \\ \langle [\mathsf{x} + 1] := 3, ([\, \mathsf{x}\colon 37 \mid \mathsf{y}\colon 1\,], [\, 37\colon 1 \mid 38\colon 2\,])\rangle \leadsto \\ ([\, \mathsf{x}\colon 37 \mid \mathsf{y}\colon 1\,], [\, 37\colon 1 \mid 38\colon 3\,]).\end{gathered}$$

$$\begin{gathered}\langle \mathsf{x} := \mathbf{cons}(1, 2) \,;\, \mathsf{y} := [\mathsf{x} + 2] \,;\, [\mathsf{x} + 1] := 3, ([\, \mathsf{x}\colon 3 \mid \mathsf{y}\colon 4\,], [\,])\rangle \leadsto \\ \langle \mathsf{y} := [\mathsf{x} + 2] \,;\, [\mathsf{x} + 1] := 3, ([\, \mathsf{x}\colon 37 \mid \mathsf{y}\colon 4\,], [\, 37\colon 1 \mid 38\colon 2\,])\rangle \leadsto \\ \mathbf{abort}.\end{gathered}$$

## Structural Operational Semantics: Notation

We write:

- $\gamma \rightsquigarrow^* \gamma'$ iff there is a finite execution from $\gamma$ to $\gamma'$.
- $\gamma \uparrow$ iff there is an infinite execution beginning with $\gamma$.

**Proposition 2** *Our semantics is* total*:*

- *For every nonterminal configuration $\gamma$, there is at least one configuration $\gamma'$ such that $\gamma \rightsquigarrow \gamma'$, i.e., executions do not get* stuck*.*
- *Every execution sequence can be extended to a sequence that is either infinite or ends with a terminal configuration.*

*Moreover, each transition preserves the domain of the store.*

The transition relation is defined by inference rules called *transition rules.*

# Transition Rules for the Simple Imperative Language

- Assignment

$$\frac{}{\langle v := e, (s, h)\rangle \rightsquigarrow ([\, s \mid v\colon [\![e]\!]_{\mathrm{exp}} s \,], h)}$$

- **skip**

$$\frac{}{\langle \mathbf{skip}, (s, h)\rangle \rightsquigarrow (s, h)}$$

- Sequencing

$$\frac{\langle c_0, (s, h)\rangle \rightsquigarrow (s', h')}{\langle c_0 \,;\, c_1, (s, h)\rangle \rightsquigarrow \langle c_1, (s', h')\rangle}$$

$$\frac{\langle c_0, (s, h)\rangle \rightsquigarrow \langle c_0', (s', h')\rangle}{\langle c_0 \,;\, c_1, (s, h)\rangle \rightsquigarrow \langle c_0' \,;\, c_1, (s', h')\rangle}$$

$$\frac{\langle c_0, (s, h)\rangle \rightsquigarrow \mathbf{abort}}{\langle c_0 \,;\, c_1, (s, h)\rangle \rightsquigarrow \mathbf{abort}}$$

For example:

$$\langle \mathsf{x} := \mathsf{x} + 1, ([\, \mathsf{x}\colon 3 \mid \mathsf{y}\colon 7 \,], [\,])\rangle \rightsquigarrow ([\, \mathsf{x}\colon 4 \mid \mathsf{y}\colon 7 \,], [\,])$$

$$\langle \mathsf{x} := \mathsf{x} + 1 \,;\, \mathsf{y} := \mathsf{y} + \mathsf{x}, ([\, \mathsf{x}\colon 3 \mid \mathsf{y}\colon 7 \,], [\,])\rangle \rightsquigarrow \langle \mathsf{y} := \mathsf{y} + \mathsf{x}, ([\, \mathsf{x}\colon 4 \mid \mathsf{y}\colon 7 \,], [\,])\rangle$$

$$\langle (\mathsf{x} := \mathsf{x} + 1 \,;\, \mathsf{y} := \mathsf{y} + \mathsf{x}) \,;\, \mathsf{x} := \mathsf{x} + 1, ([\, \mathsf{x}\colon 3 \mid \mathsf{y}\colon 7 \,], [\,])\rangle \rightsquigarrow \langle \mathsf{y} := \mathsf{y} + \mathsf{x} \,;\, \mathsf{x} := \mathsf{x} + 1, ([\, \mathsf{x}\colon 4 \mid \mathsf{y}\colon 7 \,], [\,])\rangle$$

# Conditional and **while** Commands

- Conditional

  When $[\![b]\!]_{\text{boolexp}} s = \textbf{true}$:

$$\frac{}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, (s, h)\rangle \rightsquigarrow \langle c_0, (s, h)\rangle}$$

  When $[\![b]\!]_{\text{boolexp}} s = \textbf{false}$:

$$\frac{}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, (s, h)\rangle \rightsquigarrow \langle c_1, (s, h)\rangle}$$

- **while**

  When $[\![b]\!]_{\text{boolexp}} s = \textbf{true}$:

$$\frac{}{\langle \textbf{while } b \textbf{ do } c, (s, h)\rangle \rightsquigarrow \langle c \,;\, \textbf{while } b \textbf{ do } c, (s, h)\rangle}$$

  When $[\![b]\!]_{\text{boolexp}} s = \textbf{false}$:

$$\frac{}{\langle \textbf{while } b \textbf{ do } c, (s, h)\rangle \rightsquigarrow (s, h)}$$

# A First Attempt at Variable Declarations

- **newvar**

$$\frac{}{\langle \textbf{newvar}\ v\ \textbf{in}\ c, (s, h)\rangle \rightsquigarrow \langle c\,;\, v := n, ([\, s \mid v\colon m\,], h)\rangle}$$

where $m \in \mathbf{Z}$ and $n = s\,v$.

The problem:

- Only makes sense when $v \in \mathrm{dom}\, s$.

A solution:

- **newvar**

$$\frac{}{\langle \textbf{newvar}\ v\ \textbf{in}\ c, (s, h)\rangle \rightsquigarrow \langle c\,;\, v := n, ([\, s \mid v\colon m\,], h)\rangle}$$

when $v \in \mathrm{dom}\, s$, $m \in \mathbf{Z}$, and $n = s\,v$.

$$\frac{}{\langle \textbf{newvar}\ v\ \textbf{in}\ c, (s, h)\rangle \rightsquigarrow \langle c\,;\, \textbf{delete}\ v, ([\, s \mid v\colon m\,], h)\rangle}$$

when $v \notin \mathrm{dom}\, s$, and $m \in \mathbf{Z}$.

$$\frac{}{\langle \textbf{delete}\ v, (s, h)\rangle \rightsquigarrow (s\rceil(\mathrm{dom}(s) - v), h)}$$

when $v \in \mathrm{dom}\, s$.

More problems:

- Doesn't preserve domain of store.
- Local variables are not protected from concurrent interference.

## The Concurrency Problem

We have

$$\begin{aligned}
&\langle \textbf{newvar}\ \mathsf{x}\ \textbf{in}\ (\mathsf{x} := 2\ ;\ \mathsf{y} := \mathsf{y} \times \mathsf{x}), ([\,\mathsf{x}\colon 100 \mid \mathsf{y}\colon 200\,], h)\rangle \leadsto \\
&\langle \mathsf{x} := 2\ ;\ \mathsf{y} := \mathsf{y} \times \mathsf{x}\ ;\ \mathsf{x} := 100, ([\,\mathsf{x}\colon m \mid \mathsf{y}\colon 200\,], h)\rangle \leadsto \\
&\langle \mathsf{y} := \mathsf{y} \times \mathsf{x}\ ;\ \mathsf{x} := 100, ([\,\mathsf{x}\colon 2 \mid \mathsf{y}\colon 200\,], h)\rangle \leadsto \\
&\langle \mathsf{x} := 100, ([\,\mathsf{x}\colon 2 \mid \mathsf{y}\colon 400\,], h)\rangle \leadsto \\
&([\,\mathsf{x}\colon 100 \mid \mathsf{y}\colon 400\,], h).
\end{aligned}$$

But with the concurrent composition

$$\mathsf{x} := 82 \parallel \textbf{newvar}\ \mathsf{x}\ \textbf{in}\ (\mathsf{x} := 2\ ;\ \mathsf{y} := \mathsf{y} \times \mathsf{x}),$$

interleaving will cause $\mathsf{x}{:=}82$ to affect the local variable $x$ rather than the global variable.

# The Solution for Variable Declarations

- **newvar**

$$\frac{}{\langle \textbf{newvar}\ v\ \textbf{in}\ c, (s,h)\rangle \rightsquigarrow \langle \textbf{newvar}\ v := m\ \textbf{in}\ c, (s,h)\rangle,}$$

where $m \in \mathbf{Z}$.

$$\frac{\langle c, ([\, s \mid v\colon [\![e]\!]_{\text{exp}} s\,], h)\rangle \rightsquigarrow (s', h')}{\langle \textbf{newvar}\ v := e\ \textbf{in}\ c, (s,h)\rangle \rightsquigarrow (\text{rst}(s', v, s), h'),}$$

$$\frac{\langle c, ([\, s \mid v\colon [\![e]\!]_{\text{exp}} s\,], h)\rangle \rightsquigarrow \langle c', (s', h')\rangle}{\begin{array}{l}\langle \textbf{newvar}\ v := e\ \textbf{in}\ c, (s,h)\rangle \rightsquigarrow \\ \qquad\qquad \langle \textbf{newvar}\ v := s'v\ \textbf{in}\ c', (\text{rst}(s', v, s), h')\rangle,\end{array}}$$

$$\frac{\langle c, ([\, s \mid v\colon [\![e]\!]_{\text{exp}} s\,], h)\rangle \rightsquigarrow \textbf{abort}}{\langle \textbf{newvar}\ v := e\ \textbf{in}\ c, (s,h)\rangle \rightsquigarrow \textbf{abort},}$$

where $\text{rst}(s', v, s) = \begin{cases} [\, s' \mid v\colon sv\,] & \text{when } v \in \text{dom}\, s \\ s' \rceil (\text{dom}(s) - \{v\}) & \text{when } v \notin \text{dom}\, s. \end{cases}$

For example, from the execution of the body:

$\langle \mathsf{x} := 2\ ;\ \mathsf{y} := \mathsf{y} \times \mathsf{x}, ([\,\mathsf{x}\colon m \mid \mathsf{y}\colon 200\,], h)\rangle \rightsquigarrow$
$\langle \mathsf{y} := \mathsf{y} \times \mathsf{x}, ([\,\mathsf{x}\colon 2 \mid \mathsf{y}\colon 200\,], h)\rangle \rightsquigarrow$
$([\,\mathsf{x}\colon 2 \mid \mathsf{y}\colon 400\,], h).$

we can infer

$\langle \textbf{newvar}\ \mathsf{x}\ \textbf{in}\ (\mathsf{x} := 2\ ;\ \mathsf{y} := \mathsf{y} \times \mathsf{x}), ([\,\mathsf{x}\colon 100 \mid \mathsf{y}\colon 200\,], h)\rangle \rightsquigarrow$
$\langle \textbf{newvar}\ \mathsf{x} := m\ \textbf{in}\ (\mathsf{x} := 2\ ;\ \mathsf{y} := \mathsf{y} \times \mathsf{x}), ([\,\mathsf{x}\colon 100 \mid \mathsf{y}\colon 200\,], h)\rangle \rightsquigarrow$
$\langle \textbf{newvar}\ \mathsf{x} := 2\ \textbf{in}\ (\mathsf{y} := \mathsf{y} \times \mathsf{x}), ([\,\mathsf{x}\colon 100 \mid \mathsf{y}\colon 200\,], h)\rangle \rightsquigarrow$
$([\,\mathsf{x}\colon 100 \mid \mathsf{y}\colon 400\,], h).$

# Transition Rules for the New Commands

- Allocation

$$\frac{}{\begin{array}{l}\langle v := \mathbf{cons}(e_0, \ldots, e_{n-1}), (s, h)\rangle \\ \quad \rightsquigarrow ([\, s \mid v{:}\, \ell\,], [\, h \mid \ell{:}\, [\![e_0]\!]_{\text{exp}} s \mid \ldots \mid \ell + n - 1{:}\, [\![e_{n-1}]\!]_{\text{exp}} s\,])\end{array}}$$

  where $\ell, \ldots, \ell + n - 1 \in \text{Locations} - \operatorname{dom} h$.

- Lookup

  When $[\![e]\!]_{\text{exp}} s \in \operatorname{dom} h$:

$$\frac{}{\langle v := [e], (s, h)\rangle \rightsquigarrow ([\, s \mid v{:}\, h([\![e]\!]_{\text{exp}} s)\,], h)}$$

  When $[\![e]\!]_{\text{exp}} s \notin \operatorname{dom} h$:

$$\frac{}{\langle v := [e], (s, h)\rangle \rightsquigarrow \mathbf{abort}}$$

# More New Commands

- Mutation

  When $[\![e]\!]_{\text{exp}}s \in \operatorname{dom} h$:

$$\frac{}{\langle [e] := e', (s,h)\rangle \rightsquigarrow (s, [\, h \mid [\![e]\!]_{\text{exp}}s\colon [\![e']\!]_{\text{exp}}s \,])}$$

  When $[\![e]\!]_{\text{exp}}s \notin \operatorname{dom} h$:

$$\frac{}{\langle [e] := e', (s,h)\rangle \rightsquigarrow \textbf{abort}}$$

- Disposal

  When $[\![e]\!]_{\text{exp}}s \in \operatorname{dom} h$:

$$\frac{}{\langle \textbf{dispose}\ e, (s,h)\rangle \rightsquigarrow (s, h\rceil(\operatorname{dom} h - \{[\![e]\!]_{\text{exp}}s\}))}$$

  When $[\![e]\!]_{\text{exp}}s \notin \operatorname{dom} h$:

$$\frac{}{\langle \textbf{dispose}\ e, (s,h)\rangle \rightsquigarrow \textbf{abort}}$$

## Binding in Variable Declarations

$$\begin{aligned}
\mathrm{FV}(\mathbf{newvar}\ v\ \mathbf{in}\ c) &= \mathrm{FV}(c) - \{v\}\\
\mathrm{FV}(\mathbf{newvar}\ v := e\ \mathbf{in}\ c) &= \mathrm{FV}(e) \cup (\mathrm{FV}(c) - \{v\})\\
(\mathbf{newvar}\ v\ \mathbf{in}\ c)/\delta &= \mathbf{newvar}\ v_{\mathrm{new}}\ \mathbf{in}\ (c/[\delta|v \to v_{\mathrm{new}}])\\
(\mathbf{newvar}\ v := e\ \mathbf{in}\ c)/\delta &= \mathbf{newvar}\ v_{\mathrm{new}} := (e/\delta)\ \mathbf{in}\ (c/[\delta|v \to v_{\mathrm{new}}])
\end{aligned}$$

where $\delta$ is $v_1 \to e_1, \ldots, v_n \to e_n$ and $v_{\mathrm{new}} \notin \mathrm{FV}(e_1) \cup \cdots \cup \mathrm{FV}(e_n)$.

## Free Assigned Variables

$$\begin{aligned}
\mathrm{FA}(c) &= \text{“The free variables assigned by } c\text{”}\\
\mathrm{FA}(v := e) &= \{v\}\\
\mathrm{FA}(v := \mathbf{cons}(e_0, \ldots, e_{n-1})) &= \{v\}\\
\mathrm{FA}(v := [e]) &= \{v\}\\
\mathrm{FA}(\mathbf{skip}) &= \{\}\\
\mathrm{FA}([e] := e') &= \{\}\\
\mathrm{FA}(\mathbf{dispose}\ e) &= \{\}\\
\mathrm{FA}(c_0\ ;\ c_1) &= \mathrm{FA}(c_0) \cup \mathrm{FA}(c_1)\\
\mathrm{FA}(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1) &= \mathrm{FA}(c_0) \cup \mathrm{FA}(c_1)\\
\mathrm{FA}(\mathbf{while}\ b\ \mathbf{do}\ c) &= \mathrm{FA}(c)\\
\mathrm{FA}(\mathbf{newvar}\ v\ \mathbf{in}\ c) &= \mathrm{FA}(c) - \{v\}\\
\mathrm{FA}(\mathbf{newvar}\ v := e\ \mathbf{in}\ c) &= \mathrm{FA}(c) - \{v\}
\end{aligned}$$

**Proposition 3** *If $v \notin \mathrm{FA}(c)$, and either $\langle c, (s, h)\rangle \leadsto^* \langle c', (s', h')\rangle$ or $\langle c, (s, h)\rangle \leadsto^* (s', h')$, then $s'v = s\,v$.*

# The Substitution Law for Commands

**Proposition 4** *Let $c$ be a command such that $\mathrm{FV}(c) \subseteq \{v_1, \ldots, v_n\}$, and let $\delta$ be a substitution*

$$v_1 \to e_1, \ldots, v_n \to e_n$$

*such that, when $v_i \in \mathrm{FA}(c)$, $e_i$ is a variable that does not occur in any other $e_j$.*

*For any store $s$ such that $\mathrm{FV}(e_1) \cup \cdots \cup \mathrm{FV}(e_n) \subseteq \mathrm{dom}\, s$, let*

$$\widehat{s} = [\, v_1\colon [\![e_1]\!]_{\exp} s \mid \ldots \mid v_n\colon [\![e_n]\!]_{\exp} s \,].$$

*Then*

- *If $\langle c/\delta, (s,h)\rangle \rightsquigarrow (s', h')$, then $\langle c, (\widehat{s}, h)\rangle \rightsquigarrow (\widehat{s'}.h')$.*
- *If $\langle c/\delta, (s,h)\rangle \rightsquigarrow \langle \tilde{c}, (s', h')\rangle$, then there is a command $c'$ such that $\langle c, (\widehat{s}, h)\rangle \rightsquigarrow \langle c', (\widehat{s'}, h')\rangle$ and $\tilde{c} = c'/\delta$.*
- *If $\langle c/\delta, (s,h)\rangle \rightsquigarrow$ **abort**, then $\langle c, (\widehat{s}, h)\rangle \rightsquigarrow$ **abort**.*

*and therefore*

- *If $\langle c/\delta, (s,h)\rangle \rightsquigarrow^* (s', h')$, then $\langle c, (\widehat{s}, h)\rangle \rightsquigarrow^* (\widehat{s'}.h')$.*
- *If $\langle c/\delta, (s,h)\rangle \rightsquigarrow^*$ **abort**, then $\langle c, (\widehat{s}, h)\rangle \rightsquigarrow^*$ **abort**.*
- *If $\langle c/\delta, (s,h)\rangle \uparrow$, then $\langle c, (\widehat{s}, h)\rangle \uparrow$.*

# Splitting the Heap

$$
\begin{aligned}
h_0 \perp h_1 &\text{ iff } \operatorname{dom} h_0 \cap \operatorname{dom} h_1 = \{\} \\
\operatorname{dom}(h_0 \cdot h_1) &= \operatorname{dom} h_0 \cup \operatorname{dom} h_1 \\
(h_0 \cdot h_1)\ell &= \begin{cases} h_0\ell & \text{when } \ell \in \operatorname{dom} h_0 \\ h_1\ell & \text{when } \ell \in \operatorname{dom} h_1. \end{cases}
\end{aligned}
$$

**Proposition 5** *Suppose $h = h_0 \cdot h_1$, where $h_0 \perp h_1$. Then*

- *If $\langle c, (s, h)\rangle \rightsquigarrow$ **abort**, then $\langle c, (s, h_0)\rangle \rightsquigarrow$ **abort**.*
- *If $\langle c, (s, h)\rangle \rightsquigarrow \langle c', (s', h')\rangle$ then $\langle c, (s, h_0)\rangle \rightsquigarrow$ **abort** or $\langle c, (s, h_0)\rangle \rightsquigarrow \langle c', (s', h'_0)\rangle$, where $h'_0 \perp h_1$ and $h' = h'_0 \cdot h_1$.*
- *If $\langle c, (s, h)\rangle \rightsquigarrow (s', h')$ then $\langle c, (s, h_0)\rangle \rightsquigarrow$ **abort** or $\langle c, (s, h_0)\rangle \rightsquigarrow (s', h'_0)$, where $h'_0 \perp h_1$ and $h' = h'_0 \cdot h_1$.*

*and therefore*

- *If $\langle c, (s, h)\rangle \rightsquigarrow^*$ **abort**, then $\langle c, (s, h_0)\rangle \rightsquigarrow^*$ **abort**.*
- *If $\langle c, (s, h)\rangle \rightsquigarrow^* (s', h')$ then $\langle c, (s, h_0)\rangle \rightsquigarrow^*$ **abort** or $\langle c, (s, h_0)\rangle \rightsquigarrow^* (s', h'_0)$, where $h'_0 \perp h_1$ and $h' = h'_0 \cdot h_1$.*
- *If $\langle c, (s, h)\rangle \uparrow$ then $\langle c, (s, h_0)\rangle \rightsquigarrow^*$ **abort** or $\langle c, (s, h_0)\rangle \uparrow$.*

# Syntax of Assertions

$$
\begin{aligned}
\langle \mathrm{assert} \rangle ::= \; & \langle \mathrm{boolexp} \rangle \\
& \mid \; \neg \langle \mathrm{assert} \rangle \mid \langle \mathrm{assert} \rangle \wedge \langle \mathrm{assert} \rangle \mid \langle \mathrm{assert} \rangle \vee \langle \mathrm{assert} \rangle \\
& \mid \langle \mathrm{assert} \rangle \Rightarrow \langle \mathrm{assert} \rangle \mid \langle \mathrm{assert} \rangle \Leftrightarrow \langle \mathrm{assert} \rangle \\
& \mid \forall \langle \mathrm{var} \rangle .\ \langle \mathrm{assert} \rangle \mid \exists \langle \mathrm{var} \rangle .\ \langle \mathrm{assert} \rangle \\
& \mid \langle \mathrm{exp} \rangle \mapsto \langle \mathrm{exp} \rangle \\
& \mid \mathbf{emp} \\
& \mid \langle \mathrm{assert} \rangle \ * \ \langle \mathrm{assert} \rangle \\
& \mid \langle \mathrm{assert} \rangle \mathrel{-\!\!*} \langle \mathrm{assert} \rangle
\end{aligned}
$$

# The Semantics of Assertions

$$[\![-]\!]_{\text{assert}} \in \langle\text{assert}\rangle \to \text{Stores} \to \text{Heaps} \to \mathbf{B},$$

or more precisely,

$$[\![p \in \langle\text{assert}\rangle]\!]_{\text{assert}} \in \Big(\bigcup_{V \overset{\text{fin}}{\supseteq} \text{FV}(p)} \text{Stores}_V\Big) \to \text{Heaps} \to \mathbf{B}.$$

$$\begin{aligned}
[\![b]\!]_{\text{assert}} s\, h \ &\text{iff}\ [\![b]\!]_{\text{boolexp}} s, \\
[\![\neg\, p]\!]_{\text{assert}} s\, h \ &\text{iff}\ \neg[\![p]\!]_{\text{assert}} s\, h, \\
[\![p_0 \wedge p_1]\!]_{\text{assert}} s\, h \ &\text{iff}\ [\![p_0]\!]_{\text{assert}} s\, h \wedge [\![p_1]\!]_{\text{assert}} s\, h \\
&\qquad \text{(and similarly for } \vee,\ \Rightarrow,\ \Leftrightarrow\text{)}. \\
[\![\forall v.\ p]\!]_{\text{assert}} s\, h \ &\text{iff}\ \forall x \in \mathbf{Z}.\ [\![p]\!]_{\text{assert}}([\, s \mid v\!: x\,], h), \\
[\![\exists v.\ p]\!]_{\text{assert}} s\, h \ &\text{iff}\ \exists x \in \mathbf{Z}.\ [\![p]\!]_{\text{assert}}([\, s \mid v\!: x\,], h),
\end{aligned}$$

# The Semantics of Assertions (continued)

$$
\begin{aligned}
[\![ e \mapsto e' ]\!]_{\text{assert}} s\, h \ \text{ iff} & \\
& \operatorname{dom} h = \{[\![ e ]\!]_{\exp} s\} \text{ and } h([\![ e ]\!]_{\exp} s) = [\![ e' ]\!]_{\exp} s, \\
[\![ \mathbf{emp} ]\!]_{\text{assert}} s\, h \ \text{ iff} & \ \operatorname{dom} h = \{\}, \\
[\![ p_0 \ * \ p_1 ]\!]_{\text{assert}} s\, h \ \text{ iff} & \\
& \exists h_0, h_1.\ h_0 \perp h_1 \text{ and } h_0 \cdot h_1 = h \text{ and} \\
& \qquad [\![ p_0 ]\!]_{\text{assert}} s\, h_0 \text{ and } [\![ p_1 ]\!]_{\text{assert}} s\, h_1, \\
[\![ p_0 \mathrel{-\!\!*} p_1 ]\!]_{\text{assert}} s\, h \ \text{ iff} & \\
& \forall h'.\ (h' \perp h \text{ and } [\![ p_0 ]\!]_{\text{assert}} s\, h') \text{ implies} \\
& \qquad [\![ p_1 ]\!]_{\text{assert}} s\, (h \cdot h').
\end{aligned}
$$

# Examples

$$
\begin{aligned}
[\![\mathsf{x}+\mathsf{y}=\mathsf{x}]\!]_{\text{assert}}\, s\, h \ &\text{iff}\ [\![\mathsf{x}+\mathsf{y}=\mathsf{x}]\!]_{\text{boolexp}}\, s \\
&\text{iff}\ s\,\mathsf{y}=0. \\
[\![\forall \mathsf{x}.\ \mathsf{x}+\mathsf{y}=\mathsf{x}]\!]_{\text{assert}}\, s\, h \ &\text{iff}\ \forall n \in \mathbf{Z}.\ [\![\mathsf{x}+\mathsf{y}=\mathsf{x}]\!]_{\text{assert}} [\, s \mid \mathsf{x}\colon n\,]\, h \\
&\text{iff}\ \forall n \in \mathbf{Z}.\ [\, s \mid \mathsf{x}\colon n\,]\,\mathsf{y}=0 \\
&\text{iff}\ \forall n \in \mathbf{Z}.\ s\,\mathsf{y}=0 \\
&\text{iff}\ s\,\mathsf{y}=0. \\
[\![\mathsf{x}\mapsto\mathsf{y}]\!]_{\text{assert}}\, s\, h \ &\text{iff}\ \operatorname{dom} h=\{s\,\mathsf{x}\}\ \text{and}\ h(s\,\mathsf{x})=s\,\mathsf{y} \\
&\text{iff}\ h=\{\langle s\,\mathsf{x}, s\,\mathsf{y}\rangle\}.
\end{aligned}
$$

## Some Abbreviations

$$e \mapsto - \stackrel{\text{def}}{=} \exists x'.\ e \mapsto x' \quad \text{where } x' \text{ not free in } e$$

$$e \hookrightarrow e' \stackrel{\text{def}}{=} e \mapsto e' \ * \ \mathbf{true}$$

$$e \mapsto e_0, \ldots, e_{n-1} \stackrel{\text{def}}{=} e \mapsto e_0 \ * \ \cdots \ * \ e+n-1 \mapsto e_{n-1}$$

$$e \hookrightarrow e_0, \ldots, e_{n-1} \stackrel{\text{def}}{=} e \hookrightarrow e_0 \ * \ \cdots \ * \ e+n-1 \hookrightarrow e_{n-1}$$

$$\text{iff } e \mapsto e_0, \ldots, e_{n-1} \ * \ \mathbf{true}$$

$$e \doteq e' \stackrel{\text{def}}{=} (e = e') \wedge \mathbf{emp}$$

## Examples

$$[\![\mathsf{x} \mapsto -]\!]_{\text{assert}} s\, h \ \text{ iff } \ \operatorname{dom} h = \{s\,\mathsf{x}\}$$

$$[\![\mathsf{x} \hookrightarrow \mathsf{y}]\!]_{\text{assert}} s\, h \ \text{ iff } \ s\,\mathsf{x} \in \operatorname{dom} h \text{ and } h(s\,\mathsf{x}) = s\,\mathsf{y}$$

$$[\![\mathsf{x} \hookrightarrow -]\!]_{\text{assert}} s\, h \ \text{ iff } \ s\,\mathsf{x} \in \operatorname{dom} h$$

$$[\![\mathsf{x} \mapsto \mathsf{y}, \mathsf{z}]\!]_{\text{assert}} s\, h \ \text{ iff } \ h = \{\langle s\,\mathsf{x}, s\,\mathsf{y}\rangle, \langle s\,\mathsf{x}+1, s\,\mathsf{z}\rangle\}$$

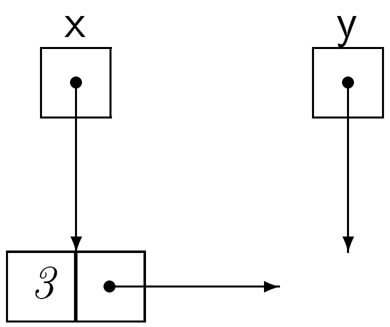
$$[\![\mathsf{x} \hookrightarrow \mathsf{y}, \mathsf{z}]\!]_{\text{assert}} s\, h \ \text{ iff } \ h \supseteq \{\langle s\,\mathsf{x}, s\,\mathsf{y}\rangle, \langle s\,\mathsf{x}+1, s\,\mathsf{z}\rangle\}$$

$$[\![e \doteq e']\!]_{\text{assert}} s\, h \ \text{ iff } \ [\![e]\!]_{\exp} s = [\![e']\!]_{\exp} s \text{ and } \operatorname{dom} h = \{\}$$

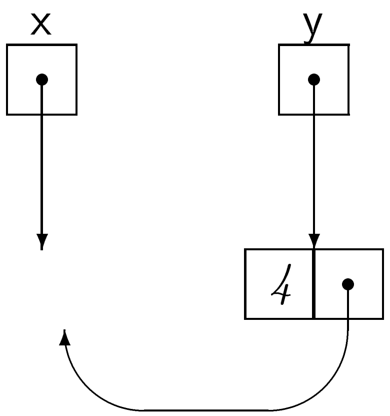
# An Example of $*$

$$[\![\mathsf{x} \mapsto 3, \mathsf{y} \ * \ \mathsf{y} \mapsto 4, \mathsf{x}]\!]_{\text{assert}} s\, h$$
$$\text{iff } \exists h_0, h_1.\ h_0 \perp h_1 \text{ and } h_0 \cdot h_1 = h$$
$$\text{and } [\![\mathsf{x} \mapsto 3, \mathsf{y}]\!]_{\text{assert}} s\, h_0 \text{ and } [\![\mathsf{y} \mapsto 4, \mathsf{x}]\!]_{\text{assert}} s\, h_1$$

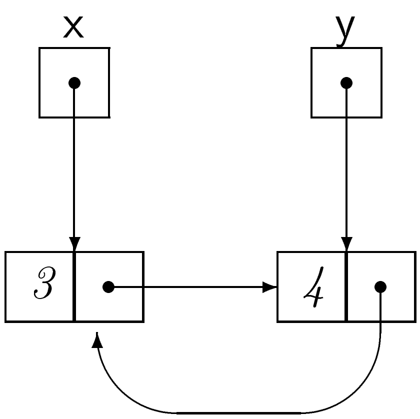
$\mathsf{x} \mapsto 3, \mathsf{y}$




$\mathsf{y} \mapsto 4, \mathsf{x}$




$\mathsf{x} \mapsto 3, \mathsf{y} \ * \ \mathsf{y} \mapsto 4, \mathsf{x}$



# More Examples of $*$

Suppose $s\,\mathsf{x}$ and $s\,\mathsf{y}$ are distinct locations, so that

$$h_1 = \{\langle s\,\mathsf{x}, 1\rangle\} \qquad \text{and} \qquad h_2 = \{\langle s\,\mathsf{y}, 2\rangle\}.$$

are heaps with disjoint domains. Then

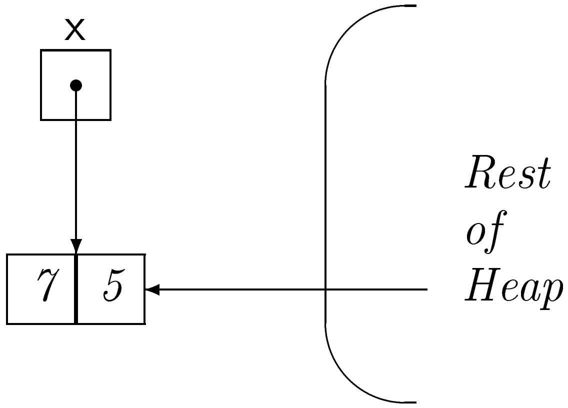
| If $P$ is: | then $[\![P]\!]s\,h$ is: |
|---|---|
| $\mathsf{x} \mapsto 1$ | $h = h_1$ |
| $\mathsf{y} \mapsto 2$ | $h = h_2$ |
| $\mathsf{x} \mapsto 1 \ * \ \mathsf{y} \mapsto 2$ | $h = h_1 \cdot h_2$ |
| $\mathsf{x} \mapsto 1 \ * \ \mathsf{x} \mapsto 1$ | **false** |
| $\mathsf{x} \mapsto 1 \vee \mathsf{y} \mapsto 2$ | $h = h_1$ or $h = h_2$ |
| $\mathsf{x} \mapsto 1 \ * \ (\mathsf{x} \mapsto 1 \vee \mathsf{y} \mapsto 2)$ | $h = h_1 \cdot h_2$ |
| $(\mathsf{x} \mapsto 1 \vee \mathsf{y} \mapsto 2) \ * \ (\mathsf{x} \mapsto 1 \vee \mathsf{y} \mapsto 2)$ | $h = h_1 \cdot h_2$ |
| $\mathsf{x} \mapsto 1 \ * \ \mathsf{y} \mapsto 2 \ * \ (\mathsf{x} \mapsto 1 \vee \mathsf{y} \mapsto 2)$ | **false** |
| $\mathsf{x} \mapsto 1 \ * \ \mathbf{true}$ | $h_1 \subseteq h$ |
| $\mathsf{x} \mapsto 1 \ * \ \neg\, \mathsf{x} \mapsto 1$ | $h_1 \subseteq h$ |

# An Example of $-\!\!*$

$$[\![\mathsf{x} \mapsto 7, 5 -\!\!* P]\!]_{\text{assert}} s\, h$$
$$\text{iff } \forall h'.\ (h' \perp h \text{ and } [\![\mathsf{x} \mapsto 7, 5]\!]_{\text{assert}} s\, h')$$
$$\text{implies } [\![P]\!]_{\text{assert}} s\, (h \cdot h')$$

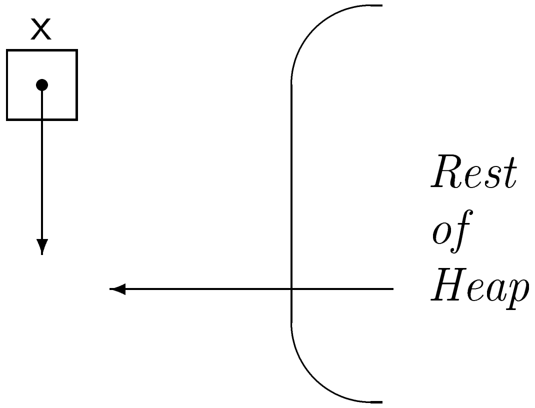
If:

$P$



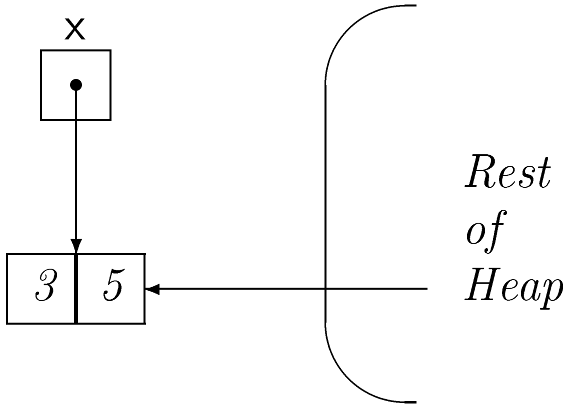

then:

$\mathsf{x} \mapsto 7, 5 -\!\!* P$




and:

$\mathsf{x} \mapsto 3, 5\ *\ (\mathsf{x} \mapsto 7, 5 -\!\!* P)$



## Some Terminology

Suppose $V \supseteq \mathrm{FV}(p)$.

- When $[\![p]\!]_{\text{assert}} s\, h = \mathbf{true}$ for some $(s, h) \in \mathrm{States}_V$, we say
  - $p$ is *true* in $(s, h)$,
  - $p$ *holds* for $(s, h)$,
  - $p$ *describes* $(s, h)$,
  - $(s, h)$ *satisfies* $p$.
- When $[\![p]\!]_{\text{assert}} s\, h = \mathbf{true}$ for all states $(s, h) \in \mathrm{States}_V$, we say
  - $p$ is *valid*.
- When $[\![p]\!]_{\text{assert}} s\, h = \mathbf{false}$ for all states $(s, h) \in \mathrm{States}_V$, we say
  - $p$ is *unsatisfiable*.
- When $[\![q]\!]_{\text{assert}} s\, h = \mathbf{true}$ holds for every state $(s, h) \in \mathrm{States}_V$ such that $[\![p]\!]_{\text{assert}} s\, h = \mathbf{true}$, we say
  - $p$ is *stronger* than $q$,
  - $q$ is *weaker* than $p$.

  (Notice that **true** is weaker than any assertion and **false** is stronger than any assertion.)
- When $[\![q]\!]_{\text{assert}} s\, h = \mathbf{true}$ holds for exactly those states $(s, h) \in \mathrm{States}_V$ such that $[\![p]\!]_{\text{assert}} s\, h = \mathbf{true}$, we say
  - $p$ is *equivalent* to $q$.

## Binding by Quantifiers

$$\begin{aligned}
\mathrm{FV}(\forall v.\ p) &= \mathrm{FV}(p) - \{v\} \\
\mathrm{FV}(\exists v.\ p) &= \mathrm{FV}(p) - \{v\} \\
(\forall v.\ p)/\delta &= \forall v_{\mathrm{new}}.\ (p/[\delta|v \to v_{\mathrm{new}}]) \\
(\exists v.\ p)/\delta &= \exists v_{\mathrm{new}}.\ (p/[\delta|v \to v_{\mathrm{new}}])
\end{aligned}$$

where $\delta$ is $v_1 \to e_1, \ldots, v_n \to e_n$ and $v_{\mathrm{new}} \notin \mathrm{FV}(e_1) \cup \cdots \cup \mathrm{FV}(e_n)$.

## The Substitution Law for Assertions

**Proposition 6** *Let $\delta$ abbreviate the substitution*

$$v_1 \to e_1, \ldots, v_n \to e_n,$$

*let $s$ be a store such that $\mathrm{FV}(e_1) \cup \cdots \cup \mathrm{FV}(e_n) \subseteq \mathrm{dom}\, s$, and let*

$$\widehat{s} = [\, v_1 \colon [\![e_1]\!]_{\mathrm{exp}} s \mid \ldots \mid v_n \colon [\![e_n]\!]_{\mathrm{exp}} s \,].$$

*If $p$ is an assertion such that $\mathrm{FV}(p) \subseteq \{v_1, \ldots, v_n\}$, then*

$$[\![p/\delta]\!]_{\mathrm{assert}} s\, h = [\![p]\!]_{\mathrm{assert}} \widehat{s}\, h.$$

Eventually, $\delta$ will map

| | | |
|---|---|---|
| ordinary variables | into | expressions |
| sequence variables | into | sequence expressions |
| tree variables | into | tree expressions |
| assertion variables | into | assertions |

$$\vdots$$

# Inference Rules

- An *inference rule* consists of zero or more *premisses* and a single *conclusion*, separated by a horizontal line:

$$\frac{\mathcal{P}_1 \qquad \cdots \qquad \mathcal{P}_n}{\mathcal{C}}$$

- The premisses and conclusion are schemata, i.e., they may contain *metavariables*, each of which ranges over some set of phrases.

- An instance of an inference rule is obtained by replacing each metavariable by a phrase in its range (satisfying any *side-conditions*). In an instance, the premisses and conclusion will be assertions (or transitions or specifications).

- A *formal proof* is a sequence of assertions, each of which is the conclusion of some instance of an inference rule whose premisses occur earlier in the sequence.

- An inference rule with zero premisses is called an *axiom schema.* The overbar is sometimes omitted.

- An axiom schema containing no metavariables is called an *axiom.* The overbar is usually omitted.

## Some Inference Rules

$$\mathsf{x} + 0 = \mathsf{x} \qquad \text{(an axiom)}$$

$$\frac{}{e_1 = e_0 \Rightarrow e_0 = e_1} \qquad \text{(an axiom schema)}$$

$$\frac{p \qquad p \Rightarrow q}{q}$$

$$\frac{p}{\forall v.\ p}$$

## Some Instances

$$\mathsf{x} + 0 = \mathsf{x}$$

$$\frac{}{\mathsf{x} + 0 = \mathsf{x} \Rightarrow \mathsf{x} = \mathsf{x} + 0}$$

$$\frac{\mathsf{x} + 0 = \mathsf{x} \qquad \mathsf{x} + 0 = \mathsf{x} \Rightarrow \mathsf{x} = \mathsf{x} + 0}{\mathsf{x} = \mathsf{x} + 0}$$

$$\frac{\mathsf{x} = \mathsf{x} + 0}{\forall \mathsf{x}.\ \mathsf{x} = \mathsf{x} + 0}$$

# A Formal Proof

| | | |
|---|---|---|
| 1. | $\mathsf{x} + 0 = \mathsf{x}$ | (First rule) |
| 2. | $\mathsf{x} + 0 = \mathsf{x} \Rightarrow \mathsf{x} = \mathsf{x} + 0$ | (Second rule) |
| 3. | $\mathsf{x} = \mathsf{x} + 0$ | (Third rule, 1, 2) |
| 4. | $\forall \mathsf{x}.\ \mathsf{x} = \mathsf{x} + 0$ | (Fourth rule, 3) |

# As a Tree

$$\dfrac{\dfrac{\dfrac{}{\mathsf{x} + 0 = \mathsf{x}} \qquad \dfrac{}{\mathsf{x} + 0 = \mathsf{x} \Rightarrow \mathsf{x} = \mathsf{x} + 0}}{\mathsf{x} = \mathsf{x} + 0}}{\forall \mathsf{x}.\ \mathsf{x} = \mathsf{x} + 0}$$

## Soundness

An inference rule is *sound* iff, for all instances,

- if the premisses of the instance are all valid (i.e. true in all states),
- then the conclusion is valid (i.e. true in all states).

(It follows that the assertions in a formal proof must all be valid.)

For example,

$$\frac{p}{\forall v.\ p}$$

is sound, since for any assertion $p$ and variable $v$, if $p$ is true in all states, then $\forall v.\ p$ is true in all states.

However,

$$\frac{}{p \Rightarrow \forall v.\ p}$$

is not sound, since its instance

$$\frac{}{\mathsf{x} > 0 \Rightarrow \forall \mathsf{x}.\ \mathsf{x} > 0}$$

has a conclusion that does not hold in a state that maps $\mathsf{x}$ into 3.

## Some Inference Rules for $\Rightarrow$ and $\neg$

$$\frac{p \qquad p \Rightarrow q}{q} \quad \text{(modus ponens)}$$

$$p \Rightarrow p$$

$$p \Rightarrow (q \Rightarrow p)$$

$$(p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r))$$

$$(\neg q \Rightarrow \neg p) \Rightarrow ((\neg q \Rightarrow p) \Rightarrow q)$$

## Other Connectives

$$(p \Leftrightarrow q) \Rightarrow (p \Rightarrow q)$$

$$(p \Leftrightarrow q) \Rightarrow (q \Rightarrow p)$$

$$(p \wedge q) \Leftrightarrow \neg(p \Rightarrow \neg q)$$

$$(p \vee q) \Leftrightarrow (\neg p) \Rightarrow q$$

$$(p \Leftrightarrow q) \Leftrightarrow (p \Rightarrow q) \wedge (q \Rightarrow p)$$

$$\mathbf{true} \Leftrightarrow (p \Rightarrow p)$$

$$\mathbf{false} \Leftrightarrow \neg\, \mathbf{true}$$

## Quantifiers

$$\frac{p}{\forall v.\ p} \quad \text{(generalization)}$$

$$(\forall v.\ p) \Rightarrow (p/v \rightarrow e)$$

$$(\forall v.\ p \Rightarrow q) \Leftrightarrow (p \Rightarrow \forall v.\ q) \qquad \text{when } v \text{ is not free in } p$$

$$(\exists v.\ p) \Leftrightarrow \neg\, (\forall v.\ \neg p)$$

$$(p/v \rightarrow e) \Rightarrow (\exists v.\ p)$$

$$(\exists v.\ p \Rightarrow q) \Leftrightarrow (p \Rightarrow \exists v.\ q) \qquad \text{when } v \text{ is not free in } p$$

## Equality

$$e = e$$
$$e = e' \Rightarrow e' = e$$
$$e = e' \wedge e' = e'' \Rightarrow e = e''$$
$$e_1 = e_2 \Rightarrow (e/v \to e_1) = (e/v \to e_2)$$

## A Derived Rule: Transitivity of Implication

$$\frac{p \Rightarrow q \qquad q \Rightarrow r}{p \Rightarrow r}$$

1. $p \Rightarrow q$ (assumption)
2. $q \Rightarrow r$ (assumption)
3. $(q \Rightarrow r) \Rightarrow (p \Rightarrow (q \Rightarrow r))$ $(p \Rightarrow (q \Rightarrow p))$
4. $p \Rightarrow (q \Rightarrow r)$ (modus ponens, 2, 3)
5. $(p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r))$
   $((p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r)))$
6. $(p \Rightarrow q) \Rightarrow (p \Rightarrow r)$ (modus ponens, 4, 5)
7. $p \Rightarrow r$ (modus ponens, 1, 6)

## Rules and Axiom Schemata for $*$ and $-\!\!*$

$$
\begin{aligned}
p_1 * p_2 &\Leftrightarrow p_2 * p_1 \\
(p_1 * p_2) * p_3 &\Leftrightarrow p_1 * (p_2 * p_3) \\
p * \mathbf{emp} &\Leftrightarrow p \\
(p_1 \vee p_2) * q &\Leftrightarrow (p_1 * q) \vee (p_2 * q) \\
(p_1 \wedge p_2) * q &\Rightarrow (p_1 * q) \wedge (p_2 * q) \\
(\exists x.\ p_1) * p_2 &\Leftrightarrow \exists x.\ (p_1 * p_2) \quad \text{when } x \text{ not free in } p_2 \\
(\forall x.\ p_1) * p_2 &\Rightarrow \forall x.\ (p_1 * p_2) \quad \text{when } x \text{ not free in } p_2
\end{aligned}
$$

$$
\frac{p_1 \Rightarrow p_2 \qquad q_1 \Rightarrow q_2}{p_1 * q_1 \Rightarrow p_2 * q_2} \quad \text{(monotonicity)}
$$

$$
\frac{p_1 * p_2 \Rightarrow p_3}{p_1 \Rightarrow (p_2 -\!\!* p_3)} \quad \text{(currying)} \qquad \frac{p_1 \Rightarrow (p_2 -\!\!* p_3)}{p_1 * p_2 \Rightarrow p_3.} \quad \text{(decurrying)}
$$

## Axiom Schemata for $\mapsto$

$$
\begin{aligned}
e_1 \mapsto e_1' \wedge e_2 \mapsto e_2' &\Leftrightarrow e_1 \mapsto e_1' \wedge e_1 = e_2 \wedge e_1' = e_2' \\
e_1 \mapsto e_1' * e_2 \mapsto e_2' * \mathbf{true} &\Rightarrow e_1 \neq e_2 \\
\mathbf{emp} &\Leftrightarrow \forall x.\ \neg(x \hookrightarrow -) \\
(e \hookrightarrow e') \wedge p &\Rightarrow (e \mapsto e') * ((e \mapsto e') -\!\!* p).
\end{aligned}
$$

# A Proof of Soundness

To show that the axiom

$$(e \hookrightarrow e') \wedge p \Rightarrow (e \mapsto e') \ * \ ((e \mapsto e') \mathrel{-\!\!*} p)$$

is sound, suppose $[\![(e \hookrightarrow e') \wedge p]\!]s, h$. Then:

- $[\![e \mapsto e' \ * \ \mathbf{true}]\!]s, h$ and $[\![p]\!]s, h$.
- There are heaps $h_0$ and $h_1$ such that $h_0 \perp h_1$, $h_0 \cdot h_1 = h$, and $[\![e \mapsto e']\!]s, h_0$.
- Thus $h_0 = [\, [\![e]\!]s \colon [\![e']\!]s \,]$.
- To see that $[\![(e \mapsto e') \mathrel{-\!\!*} p]\!]s, h_1$, let $h'$ be any heap such that $h' \perp h_1$ and $[\![e \mapsto e']\!]s, h'$. Then $h' = [\, [\![e]\!]s \colon [\![e']\!]s \,] = h_0$, and $[\![p]\!]s, h' \cdot h_1 = [\![p]\!]s, h$ is true.
- Then $[\![(e \mapsto e') \ * \ ((e \mapsto e') \mathrel{-\!\!*} p)]\!]s, h_0 \cdot h_1 = [\![(e \mapsto e') \ * \ ((e \mapsto e') \mathrel{-\!\!*} p)]\!]s, h$ is true.

# Pure Assertions

An assertion is *pure* if it does not contain $\mapsto$ or **emp**.

A pure assertion does not depend upon the heap.

# Axiom Schemata for Purity

$$p_1 \wedge p_2 \Rightarrow p_1 * p_2 \quad \text{when } p_1 \text{ or } p_2 \text{ is pure}$$

$$p_1 * p_2 \Rightarrow p_1 \wedge p_2 \quad \text{when } p_1 \text{ and } p_2 \text{ are pure}$$

$$(p \wedge q) * r \Leftrightarrow (p * r) \wedge q \qquad \text{when } q \text{ is pure}$$

$$(p_1 \mathrel{-\!\!*} p_2) \Rightarrow (p_1 \Rightarrow p_2) \quad \text{when } p_1 \text{ is pure}$$

$$(p_1 \Rightarrow p_2) \Rightarrow (p_1 \mathrel{-\!\!*} p_2) \quad \text{when } p_1 \text{ and } p_2 \text{ are pure}$$

# Two Unsound Axiom Schemata

$$p \not\Rightarrow p \ast p \qquad \text{(Contraction)}$$

e.g. $p : \mathsf{x} \mapsto 1$

$$p \ast q \not\Rightarrow p \qquad \text{(Weakening)}$$

e.g. $p : \mathsf{x} \mapsto 1$

$q : \mathsf{y} \mapsto 2$

# Some Derived Inference Rules

$$\frac{}{q \ast (q \mathrel{-\!\!\ast} p) \Rightarrow p}$$

1. $q \ast (q \mathrel{-\!\!\ast} p) \Rightarrow (q \mathrel{-\!\!\ast} p) \ast q$ $\quad (p_1 \ast p_2 \Rightarrow p_2 \ast p_1)$
2. $(q \mathrel{-\!\!\ast} p) \Rightarrow (q \mathrel{-\!\!\ast} p)$ $\quad (p \Rightarrow p)$
3. $(q \mathrel{-\!\!\ast} p) \ast q \Rightarrow p$ $\quad$ (decurrying, 2)
4. $q \ast (q \mathrel{-\!\!\ast} p) \Rightarrow p$ $\quad$ (trans impl, 1, 3)

$$\frac{}{r \Rightarrow (q \mathrel{-\!\!\ast} (q \ast r))}$$

1. $(r \ast q) \Rightarrow (q \ast r)$ $\quad (p_1 \ast p_2 \Rightarrow p_2 \ast p_1)$
2. $r \Rightarrow (q \mathrel{-\!\!\ast} (q \ast r))$ $\quad$ (currying, 1)

$$\frac{}{(p \ast r) \Rightarrow (p \ast (q \mathrel{-\!\!\ast} (q \ast r)))}$$

1. $p \Rightarrow p$ $\quad (p \Rightarrow p)$
2. $r \Rightarrow (q \mathrel{-\!\!\ast} (q \ast r))$ $\quad$ (derived above)
3. $(p \ast r) \Rightarrow (p \ast (q \mathrel{-\!\!\ast} (q \ast r)))$ $\quad$ (monotonicity, 1, 2)

# More Derived Inference Rules

$$\frac{p_1 \Rightarrow (q \mathrel{-\!\!*} r) \qquad p_2 \Rightarrow (r \mathrel{-\!\!*} s)}{p_2 \,*\, p_1 \Rightarrow (q \mathrel{-\!\!*} s)}$$

| | | |
|---|---|---|
| 1. | $p_2 \Rightarrow p_2$ | $(p \Rightarrow p)$ |
| 2. | $p_1 \Rightarrow (q \mathrel{-\!\!*} r)$ | (assumption) |
| 3. | $p_1 \,*\, q \Rightarrow r$ | (decurrying, 2) |
| 4. | $p_2 \,*\, p_1 \,*\, q \Rightarrow p_2 \,*\, r$ | (monotonicity, 1, 3) |
| 5. | $p_2 \Rightarrow (r \mathrel{-\!\!*} s)$ | (assumption) |
| 6. | $p_2 \,*\, r \Rightarrow s$ | (decurrying, 5) |
| 7. | $p_2 \,*\, p_1 \,*\, q \Rightarrow s$ | (trans impl, 4, 6) |
| 8. | $p_2 \,*\, p_1 \Rightarrow (q \mathrel{-\!\!*} s)$ | (currying, 7) |

## More Derived Inference Rules

$$\frac{p' \Rightarrow p \qquad q \Rightarrow q'}{(p \mathrel{-\!\!*} q) \Rightarrow (p' \mathrel{-\!\!*} q').}$$

| | | |
|---|---|---|
| 1. | $(p \mathrel{-\!\!*} q) \Rightarrow (p \mathrel{-\!\!*} q)$ | $(p \Rightarrow p)$ |
| 2. | $p' \Rightarrow p$ | (assumption) |
| 3. | $(p \mathrel{-\!\!*} q) * p' \Rightarrow (p \mathrel{-\!\!*} q) * p$ | (monotonicity, 1, 2) |
| 4. | $(p \mathrel{-\!\!*} q) * p \Rightarrow q$ | (decurrying, 1) |
| 5. | $(p \mathrel{-\!\!*} q) * p' \Rightarrow q$ | (trans impl, 3, 4) |
| 6. | $q \Rightarrow q'$ | (assumption) |
| 7. | $(p \mathrel{-\!\!*} q) * p' \Rightarrow q'$ | (trans impl, 5, 6) |
| 8. | $(p \mathrel{-\!\!*} q) \Rightarrow (p' \mathrel{-\!\!*} q')$ | (currying, 7) |