Finora ci siamo occupati del problema di stabilire la correttezza di un programma, o di un frammento di programma, e abbiamo formalizzato la nozione di correttezza attraverso le triple di Hoare: fissata una precondizione p e una postcondizione q, diciamo che un comando C è corretto rispetto a p e q se la tripla $\{p\}C\{q\}$ è valida, cioè se ogni volta che si esegue il comando C a partire da uno stato in cui vale p, si raggiunge uno stato in cui vale q (abbiamo distinto la correttezza parziale dalla correttezza totale).

Abbiamo poi sviluppato un calcolo logico, cioè un insieme di regole di derivazione che ci permettono di derivare, o dimostrare, una tripla. Si può dimostrare che il calcolo è corretto: ogni tripla derivabile è valida; si può anche dimostrare che il calcolo è relativamente completo: ogni tripla valida è derivabile, prendendo come assiomi tutte le formule vere dell'aritmetica (perciò non possiamo dire che il calcolo è completo in termini assoluti).

Ora cambiamo prospettiva, muovendoci però nello stesso contesto formale. Anziché partire da una tripla, e cercare di derivarla, partiamo da una coppia formata da un comando e da una formula, che interpretiamo come la postcondizione che ci aspettiamo valga dopo l'esecuzione del comando.

Ci possiamo allora chiedere quali precondizioni siano compatibili con l'obiettivo dato. In altre parole, data la coppia C $\{q\}$, per quali precondizioni la tripla $\{p\}$ C $\{q\}$ è valida (e derivabile)?

Partiamo da un esempio banale. Consideriamo il comando y := 2*x + 1 e la postcondizione $y > x$. La precondizione $x = 10$ è senz'altro una soluzione del problema. D'altra parte, anche $x = 1$ e $x \geq 1$ sono soluzioni, ed è facile trovarne altre. Possiamo determinare quale sia la soluzione migliore? Esiste un criterio ragionevole per confrontare due soluzioni, e stabilire se una delle due è migliore dell'altra?

Cercheremo di arrivare a una risposta seguendo un ragionamento generale. Questo ragionamento parte dalla nozione di stato di un programma, che abbiamo già definito e usato.

Sia V l'insieme delle variabili che compaiono nel comando C. Definiamo l'insieme Σ delle funzioni da V a $\mathbb{Z}$.

$$\Sigma = \{\sigma \mid \sigma : V \to \mathbb{Z}\}$$

(Adoprerò anche la notazione $\mathbb{Z}^V$ per indicare l'insieme di tutte le funzioni da V a $\mathbb{Z}$).

Queste funzioni somigliano molto agli stati possibili del programma; se ne discostano solo per la mancanza del valore del program counter. Possiamo considerare Σ come l'insieme degli stati iniziali potenziali di C.

Indichiamo con Π l'insieme di tutte le formule costruite su V. Sappiamo che cosa significa l'asserzione "la formula p è valida in σ", che denotiamo $\sigma \models p$ (diciamo anche che p è vera, o verificata, in σ, oppure che σ soddisfa p).

Il simbolo $\models$ rappresenta quindi una relazione binaria fra Σ e Π. Da una relazione di questo tipo, possiamo derivare diverse costruzioni insiemistiche. Qui ce ne interessano due in particolare: a ogni stato associamo l'insieme delle formule che sono valide in quello stato; a ogni formula associamo l'insieme degli stati che la soddisfanno. Formalmente, scriviamo (per ogni stato σ e per ogni formula p)

$$t(\sigma) = \{p \in \Pi \mid \sigma \models p\}$$

e

$$m(p) = \{\sigma \in \Sigma \mid \sigma \models p\}$$

Le due funzioni t e m si possono estendere a insiemi, rispettivamente di stati e di formule. A un insieme S di stati associamo l'insieme delle formule che sono vere in ogni elemento di S; a ogni insieme F di formule associamo l'insieme degli stati che le soddisfano tutte:

$$t(S) = \{p \in \Pi \mid \forall s \in S : s \models p\} = \bigcap_{s \in S} t(s)$$

e

$$m(F) = \{s \in \Sigma \mid \forall p \in F : s \models p\} = \bigcap_{p \in F} m(p)$$

ESERCIZIO Siano A e B due sottoinsiemi di formule. Dimostrare che, se $A \subseteq B$, allora $m(B) \subseteq m(A)$.

Fissiamo ora due formule, p e q. Possiamo usarle per costruire nuove formule, ad esempio $\neg p$, $p \vee q$, $p \wedge q$, $p \to q$. C'è una relazione tra gli insiemi di stati associati a p e a q e quelli associati alle formule derivate?

Cominciamo dal caso più semplice: $\neg p$. Poiché trattiamo con la logica proposizionale classica, valgono il principio del terzo escluso (le formule del tipo $p \vee \neg p$ sono tutte tautologie) e il principio di non contraddizione (le formule del tipo $p \wedge \neg p$ sono tutte false).

Quindi, in uno stato σ deve valere esattamente una fra p e $\neg p$. Concludiamo allora che $m(p) = \Sigma \setminus m(\neg p)$.

Consideriamo ora $p \wedge q$. Uno stato σ soddisfa questa formula se soddisfa sia p sia q. È facile concludere che $m(p \wedge q) = m(p) \cap m(q)$. In altre parole, il connettivo logico di congiunzione si traduce nell'operazione di intersezione. In modo analogo, si stabilisce che $m(p \vee q) = m(p) \cup m(q)$.

Il caso dell'implicazione è più interessante. Il connettivo logico di implicazione, infatti, ha una duplice natura: da un lato è, appunto un connettivo binario che serve a costruire nuove formule a partire da formule date; in questo senso, nella logica classica, si può definire a partire da negazione e disgiunzione: $p \to q \equiv (\neg p \vee q)$, e quindi è facile ricavare la relazione

$$m(p \to q) = (\Sigma \setminus m(p)) \cup m(q)$$

D'altro lato, però, l'implicazione esprime anche una relazione fra due formule. Nel nostro contesto, possiamo esprimere questa relazione dicendo che, se p implica q, allora ogni stato che soddisfa p deve soddisfare anche q (noto di passaggio che questa osservazione è alla base della regola di derivazione della conseguenza). Possiamo allora dedurre che, se p implica q, allora $m(p) \subseteq m(q)$. Da un certo punto di vista, questo significa che p dà un'informazione più precisa sullo stato del programma. Diremo che q è *più debole* di p.

Torniamo all'esempio da cui eravamo partiti. Dato il comando `y := 2*x + 1` e la postcondizione $y > x$, abbiamo individuato due possibili precondizioni: $p \equiv x > 10$ e $q \equiv x \geq 1$. Osserviamo che p implica q. In generale, date due precondizioni accettabili, non è detto che una implichi l'altra. Ad esempio, $x = 2$ e $x = 1$ sono entrambe accettabili, ma nessuna implica l'altra.

Esiste, in questo esempio, una precondizione "migliore" di tutte? Un possibile criterio è questo: la precondizione ottima è quella che impone meno vincoli sullo stato iniziale del programma, cioè quella meno restrittiva o, in altre parole, la più debole (in inglese, la *weakest precondition*).

Nel caso in esame, è facile scoprire che la precondizione più debole è $p_0 \equiv x \geq 0$ (ricordate che le variabili sono tutte di tipo *intero*): ogni stato iniziale per cui $y > x$ è valida dopo l'assegnamento deve soddisfare p_0.

Osserviamo che p_0 è proprio la precondizione che si ottiene applicando la regola di derivazione dell'assegnamento. Da questo punto di vista, la regola dell'assegnamento è speciale, perché si può dimostrare che la sua applicazione fornisce sempre la precondizione più debole.

Nel caso di un comando generico, prima di chiedersi quale sia la precondizione più debole, occorre dimostrare che esista sempre. La dimostrazione esula dagli scopi di questo corso, e la daremo per assodata.

Possiamo ora enunciare un problema interessante:

> dati un comando C e una formula q sulle variabili di C, determinare la precondizione più debole p per la quale $\{p\}\ C\ \{q\}$ è una tripla valida.

Spesso la soluzione di questo problema è indicata dalla notazione $\mathrm{wp}(C, q)$. Abbiamo già visto che la regola di derivazione per l'assegnamento ci dà an-

che la soluzione al problema di determinare $\mathrm{wp}(C, q)$ quando C consiste in un assegnamento.

Partendo da questo caso, possiamo determinare la precondizione più debole per gli altri tipi di comando, come segue.

sequenza $\mathrm{wp}(C_1; C_2, q) = \mathrm{wp}(C_1, \mathrm{wp}(C_2, q))$

scelta se $C \equiv$ `if B then F else D fi`, allora $\mathrm{wp}(C, q) = (B \wedge \mathrm{wp}(F, q)) \vee (\neg B \wedge \mathrm{wp}(D, q))$

iterazione se $C \equiv$ `while B do S od`, allora $\mathrm{wp}(C, q) = (\neg B \wedge q) \vee (B \wedge \mathrm{wp}(S; C, q))$

L'ultima clausola merita un commento a parte. Quando il comando è un'istruzione di iterazione, la precondizione più debole è data da una disgiunzione, che corrisponde a due possibili casi: (1) la condizione di ciclo B è falsa nello stato iniziale; in questo caso, nessuna istruzione viene eseguita, quindi q dev'essere già vera prima dell'esecuzione; (2) il ciclo viene eseguito almeno una volta; in questo caso, B è vera nello stato iniziale, e C è equivalente al programma formato da un'esecuzione di S seguita da C stesso.

La natura ricorsiva di questa definizione impedisce di tradurla in una procedura automatica costruttiva di calcolo. Si può comunque dimostrare che la definizione è corretta.

Il paragrafo 15.5 del libro di Ben-Ari contiene esempi di determinazione di $\mathrm{wp}(C, q)$.