Given a state consisting of one highly classified and one lowly classified object:

\[
\text{\underline{STATE}[p]} \quad \text{\underline{high, low : bool}}
\]

can we specify loosely an operation on the state which does not result in any information transfer from ‘high’ to ‘low’?

\[
\Delta \text{STATE}[p] \quad \text{\underline{STATE, STATE’}}
\]

It is easy enough to give a specific operation satisfying this requirement, but to capture the requirement loosely we have to use a loose specification outside of the schema, e.g.:
\[ f: \text{STATE} \rightarrow \text{STATE} \]

\[ \forall s^1, s^2: \text{STATE} . (s^1.\text{low} = s^2.\text{low}) \Rightarrow (f s^1).\text{low} = (f s^2).\text{low} \]

We could then write our schema:

\[ \Delta_{\text{STATE}[p]} \]

\[ \text{STATE, STATE'} \]

\[ \theta_{\text{STATE'}} = f \theta_{\text{STATE}} \]

but since all the work has been done in the specification of ‘f’ the use of the schema appears superfluous.
Note that in the axiomatic definition of \( f \), the requirement is expressed as a property of \( f \), but this property has not itself been given a name.

It is therefore not possible to express in the object language the claim that some other explicitly defined function has this property.

For example the following function has the required property:

\[
g : \text{STATE} \rightarrow \text{STATE} \\
\forall s : \text{STATE} \cdot f s = s
\]

but we cannot state this in Z without restating the original property (though it can be said in the metalanguage).
To enable such correctness propositions to be expressed we must give a name to the property itself as follows:

\[ \text{secure} : \mathbb{P} (\text{STATE} \rightarrow \text{STATE}) \]

\[ f \in \text{secure} \iff \forall s^1 s^2 : \text{STATE} \cdot (s^1.\text{low} = s^2.\text{low}) \Rightarrow (f s^1).\text{low} = (f s^2).\text{low} \]

The conjecture that ‘g’ satisfies this specification can now be expressed:

\[ \vdash g \in \text{secure} \]
If we define a further requirement:

\[
\text{safe : } \mathcal{P} (\text{STATE} \rightarrow \text{STATE})
\]

\[
f \in \text{safe} \iff \\
\forall s^1, s^2:\text{STATE} \bullet (s^1.\text{high} = s^2.\text{high}) \\
\Rightarrow (f s^1).\text{high} = (f s^2).\text{high}
\]

Then the combination of these two requirements:

\[
\text{no_flow : } \mathcal{P} (\text{STATE} \rightarrow \text{STATE})
\]

\[
\text{no_flow} = \text{secure} \cap \text{safe}
\]

may be regarded as a REFINEMENT of the original specification "secure".

That it is a refinement can be expressed in the object language as the conjecture:

\[
\vdash \text{no_flow} \subseteq \text{secure}
\]

Note that here refinement is defined as a relationship between specifications which is distinct from the relationship between a specification and an implementation.
SPECIFYING OPERATIONS AS FUNCTIONS

Type of Object
AUTO

Type of Specification
Π AUTO

Type of Operation
\[ \text{IN} \times \text{STATE} \rightarrow \text{STATE} \times \text{OUT} \]
\[ \subseteq \Pi(\text{IN} \times \text{STATE} \times \text{STATE} \times \text{OUT}) \]

Type of Specification of Operation
\[ \Pi (\text{IN} \times \text{STATE} \rightarrow \text{STATE} \times \text{OUT}) \]

Type of Non-Deterministic Operation
\[ \text{IN} \times \text{STATE} \rightarrow \Pi^1 (\text{STATE} \times \text{OUT}) \]

Type of Specification of Non-Deterministic Operation
\[ \Pi (\text{IN} \times \text{STATE} \rightarrow \Pi^1 (\text{STATE} \times \text{OUT})) \]

Type of Partial Operation
\[ \text{IN} \times \text{STATE} \nrightarrow \text{STATE} \times \text{OUT} \]

Type of Specification of Partial Operation
\[ \Pi (\text{IN} \times \text{STATE} \nrightarrow \text{STATE} \times \text{OUT}) \]

Type of Partial Non-Deterministic Operation
\[ \text{IN} \times \text{STATE} \nrightarrow \Pi^1 (\text{STATE} \times \text{OUT}) \]

Type of Specification of Partial Non-Deterministic Operation
\[ \Pi (\text{IN} \times \text{STATE} \nrightarrow \Pi^1 (\text{STATE} \times \text{OUT})) \]
Z SCHEMAS INTERPRETED AS OPERATIONS

Until the publication of Spivey’s book "understanding Z" no account was available of how schemas are to be interpreted as specifications of operations.

Spivey gives an account of a satisfaction relationship between schemas and implementations which can be formalised within Z as follows.

Let us consider this with reference to schemas describing the secure operations discussed above.

The type of a schema describing an operation over STATE is:

\[
\text{SOPTYPE} == \text{IP} \Delta \text{STATE}
\]

According to Spivey this is a loose specification of a non-deterministic partial operation (in the general case). It might therefore be re-represented as an entity of type:

\[
\text{FTYPE} == \text{IN} \times \text{STATE} \rightarrow \text{IP}^1 (\text{STATE} \times \text{OUT})
\]

\[
\text{INTTYPE} == \text{IP} (\text{IN} \times \text{STATE} \rightarrow \text{IP}^1 (\text{STATE} \times \text{OUT}))
\]
A formal account of this interpretation would therefore be a map from OPTYPE to INTTYPE:

\[
\text{MAPTYPE} = \text{OPTYPE} \rightarrow \text{INTTYPE}
\]

\[
\text{\_satisfies\_: FTYPE} \leftrightarrow \text{SOPTYPE}
\]

\[
\forall \text{SOPTYPE:S, FTYPE:f}\cdot \\
\text{f satisfies S} \iff \\
\text{dom f} = \{ \text{S} \cdot \theta \text{STATE} \} \\
\land \forall s:\text{STATE} \mid s \in \text{dom f}\cdot \\
\text{f s} \subseteq \{ \text{S} \mid \theta \text{STATE} = s \cdot \theta \text{STATE}' \}
\]

note that:

\[
\text{f satisfies S} \land \text{g satisfies S} \\
\implies (\text{f merge g}) \text{ satisfies S}
\]

where \((\text{f merge g}) \, x = f \, x \cup g \, x\)