Universität Augsburg

Relational Geometry Modelling Execution of Structured Programs

Bernhard Möller, Tony Hoare

Report 2022-03 October 2022

Institut für Informatik
D-86135 Augsburg

Relational GeometryModelling Execution of Structured Programs

Bernhard Möller[1] and Tony Hoare[2]

[1] Universität Augsburg
[2] University of Cambridge

Abstract. We discuss some twists around Concurrent Kleene Algebra (CKA). First, a new model of CKA represents a trace of a concurrent program as a diagram in a two-dimensional non-metric finite geometry, namely, program actions by points, objects and threads by vertical lines, transactions by horizontal lines, communications and resource sharing by sloping lines. While we had already sketched this earlier, we fully formalise it here in terms of the algebra of binary relations. Second, we present a new definition technique for partial operators, namely an assume/claim style akin to rely/guarantee program specification. This admits a general refinement order with Top and Bottom as well as proofs of the CKA laws. Finally, we give a short perspective on the geometric representation of some standard concurrent programming concepts.

1 Introduction

A trace of the execution of a concurrent object-oriented program can be displayed in two dimensions as a diagram of a non-metric finite geometry. The actions of the program are represented by points, its objects and threads by vertical lines, its transactions by horizontal lines, its communications and resource sharing by sloping arrows, and its partial traces by rectangular figures. These observations went into the predecessor paper [16], where it was shown semi-formally that the geometry satisfies the laws of Concurrent Kleene Algebra (CKA). In the present paper we give a formalisation of the mentioned geometry in terms of the algebra of binary relations; on this basis the CKA laws can be proved fully formally. These laws describe and justify the interleaved implementation of multithreaded programs on computer systems with a lesser number of concurrent processors. More familiar forms of semantics (e.g., verification-oriented and operational) can be derived from CKA. Programs are represented as sets of all their possible traces of execution, and non-determinism is introduced as union of these sets. The single traces are not linear sequences but rather general graphlets. The geometry is extended to multiple levels of abstraction and granularity; a method call at a higher level can be modelled by a specification of the method body, which is implemented at a lower level. The methodology is illustrated with a small-scale examples and an outlook on further applications.

Most proofs are deferred to Appendix B.

Part I: Graphlets and Geometry

All our specifications and programs will be modelled using relations with a geometric interpretation as program graphs.

2 Relational Notions

As usual, a *(binary) relation* over some set M is a subset of $M \times M$. The *identity relation* is denoted by I, the *empty relation* by $\{\}$ and the universal relation by U. Since relations are sets, we may form their union and intersection. Relational *composition* of R, R' is denoted by the juxtaposition RR'. We follow the usual convention that composition binds tighter than the set theoretic operators $\cap$ and $\cup$. Further material on relations can be found in Appendix A.

3 Graphlets

As mentioned in the introduction we use non-linear traces consisting of *events* from a set EV that may be connected by *arrows* and hence are a particular form of graphs which we call *graphlets*. The basic ideas were already sketched in [14] and studied in more detail in [20]. However, the relational formulation presented here is new. The algebra comprises operators for combining smaller graphlets into larger ones; conversely they can be used to state that a larger graph may be decomposed into smaller ones.

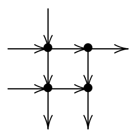
To emphasise our geometric view of traces and programs, we call events now *points*. We use binary relations $R \subseteq \mathrm{EV} \times \mathrm{EV}$ to encode the arrows and points of directed graphs. Since we do not consider graphs with self-loops on points we can represent a point a as the pair $(a, a) \in I$.

Definition 3.1. We call elements of I *points* and all other pairs *arrows*. Hence the points and arrows of a relation R are $\mathit{points}(R) = R \cap I$ and $\mathit{arrows}(R) = R - I$, where $-$ is set-theoretic difference. A relation is uniquely determined by its points and arrows, since $R = \mathit{points}(R) + \mathit{arrows}(R)$, where $+$ denotes disjoint union. To emphasize the connection with graph theory we call relations now *graphlets*. Define $\mathit{tail}(a,b) = (a,a)$, $\mathit{head}(a,b) = (b,b)$, and $\mathit{ends}(a,b) = \{\mathit{tail}(a,b), \mathit{head}(a,b)\}$. An arrow will preferably be drawn or imagined with its tail on the left (or above) and its head on the right (or below).

Let $\mathit{head}_R, \mathit{tail}_R$ be the range-restrictions of *head* and *tail* to $\mathit{points}(R)$, i.e., $\mathit{head}_R(a,b)$ is $\mathit{head}(a,b)$ when $\mathit{head}(a,b) \in \mathit{points}(R)$ and undefined otherwise, and analogously for tail_R. The quadruple $(\mathit{points}(R), \mathit{arrows}(R), \mathit{head}_R, \mathit{tail}_R)$ satisfies the standard definition of a directed graph, with the important exception that the head_R and tail_r functions may be partial rather than total functions on $\mathit{arrows}(R)$. This exception facilitates simple definitions of operators for graphlet composition.

Example 3.2.
The set *points*(R) is indicated by bullets. Those tails/heads of arrows that lie outside *points*(R) are not shown. □

$R:$



Definition 3.3. The two functions *tails*(R) and *heads*(R) are the element-wise liftings of the *head* and *tail* functions from pairs to the set R. Therefore they distribute through arbitrary unions of graphlets. Relation-algebraically they coincide with the domain and range operators; hence

$$\mathit{tails}(R) = RU \cap I \ , \qquad \mathit{heads}(R) = UR \cap I \ ,$$

where $U =_{df} \mathrm{EV}\times\mathrm{EV}$ is the universal graphlet. By this,

$$\mathit{points}(R) \subseteq \mathit{tails}(R) \cap \mathit{heads}(R) \ . \tag{1}$$

Generally equality does not hold, since arrows may lead into or out of the graph represented by graphlet R; the end points of these do not lie in *points*(R).

Remark 3.4.
Our model of graphs comprises also ones with a so-called *N shape*, i.e., a particular combination of a fork and a join. Such N shapes cannot be produced in standard pomset semantics [9] where only "context-free" sequential and parallel composition of pomsets are available. For more interesting examples we refer to [20]. A recent related but much more complex approach to N shapes was presented in [7].

$$\begin{tikzcd} a \arrow[d] \arrow[dr] & b \arrow[d] \\ c & d \end{tikzcd}$$

□

Remark 3.5. So far our arrows do not carry labels. However, frequently these are essential. Assume a set L of *labels*. Then a graphlet with labelled arrows can be modelled as a relation R such that $R = \bigcup_{l\in L} R_l$, where $(R_l)_{l\in L}$ is a family of relations. For events e, e' and label l one can then write $e \overset{l}{\to} e'$ when $(e,e') \in R_l$. In the sequel we therefore freely use labels whenever appropriate. □

4 Dependence

Let Dep be a relation, where each of its arrows represent a direct causal dependence between its tail and its head, which are actions performed by a computer in a single execution of a program.

Requirement 4.1. We stipulate that *points*$(\mathsf{Dep}) = \{(a,a) \mid a \in \mathrm{EV}\}$ and that all our graphlets are subsets of Dep.

Definition 4.2.
1. R is *point-closed* if every point which is the head of one arrow in R and the tail of another is also in R:

$$\mathit{heads}(R) \cap \mathit{tails}(R) \subseteq R \ . \tag{2}$$

2. R is *arrow-closed* if every arrow with head or tail in R is also in R.

3. R is *doubly closed* if it has both properties. Thus Dep is doubly closed.

An arrow-closed graphlet has full information about how it Dep-connects to its environment. A counterexample is the following.

Example 4.3.
Let $points(R) = \{(a,a),(b,b)\}$, $arrows(R) = \{(a,b),(a,d)\}$ and $points(S) = \{(c,c),(d,d)\}$, $arrows(S) = \{(c,b),(c,d)\}$. The arrows of R are drawn solid, the ones of S dashed. Both R and S are not arrow-closed, since the diagonal arrows violate the condition w.r.t. their heads. Adding the down-left arrow to R and the down-right arrow to S yields arrow-closed graphlets. The overall graph contains two instances of N shapes. □

$$\begin{tikzcd} a \arrow[d, "R"'] \arrow[dr] & c \arrow[dl, dashed] \arrow[d, dashed, "S"] \\ b & d \end{tikzcd}$$

To formalise arrow-closedness we use that composition with a set P of points *restricts* an arrow set A: the relation PA (AP) consists of those arrows in A whose tail (head) is in P. For graphlet R restrictions have additional effect:

$$PR = (P \cap points(R)) \cup P\, arrows(R)\ ,\ \ RP = (P \cap points(R)) \cup arrows(R)\, P\ .$$

If R is a point set Q then these formulas simplify to $PQ = P \cap Q = QP$. In particular, $PP = P$.

Now we can give a formal definition.

Definition 4.4. Relation R is *arrow-closed* if, with $P =_{df}$ $points(R)$,

$$P\,\mathsf{Dep} \cup \mathsf{Dep}\,P \subseteq R\ . \tag{3}$$

The following properties show that arrow-closed graphlets are tightly coupled.

Lemma 4.5. *Let R be arbitrary and S arrow-closed.*
1. *With $P =_{df}$ $points(R)$, $Q =_{df}$ $points(S)$ we have $PRQ \subseteq PSQ$.*
2. *Hence, if also R is arrow-closed then $PRQ = PSQ$ and $QSP = QRP$. This means that R and S agree in their interconnecting arrows.*

Proof. By Req. 4.1 of Sect. 4 we have $R \subseteq \mathsf{Dep}$ and hence Part 1 is shown by $PRQ \subseteq P\,\mathsf{Dep}\,Q = P\,\mathsf{Dep}\,QQ \subseteq PSQ$. Part 2 is immediate from that. □

For the further discussion we need additional relational notions. The basic idea of relational composition, namely to combine two short steps into one longer, can be iterated to cover arbitrarily many elementary steps. This applies, in particular, to the dependence relation Dep.

Definition 4.6. The transitive closure S^+ of relation S is defined as usual.
1. Graphlet R is *intransitive* if $(arrows(R)^+ arrows(R)^+) \cap arrows(R) = \{\}$. Then R is a parsimonious way of representing $arrows(R)^+$, since it does not contain arrows that could be inferred by transitivity. This reduces clutter in the diagrammatic representation of $arrows(R)^+$ considerably. In the literature then R is called a *Hasse diagram* for $arrows(R)^+$.
2. Graphlet R is called *acyclic* if it does not admit a proper path from any point to itself, i.e., if $arrows(R)^+ \cap I = \{\}$.

We assume Dep to be intransitive: this is what is meant by the directness of the dependence. General causal dependence is represented by the transitive closure Dep^+. We do not require Dep to be acyclic. Any cycle in Dep represents a transaction, whose actions all must occur simultaneously (or appear to do so). A programming language will usually place severe restrictions on the choice of which transactions it will accept; all other cycles will be defined as programming errors (deadlocks).

5 Interfaces

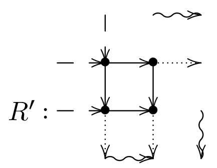
Definition 5.1. Graphlets R, R' are *point-disjoint* if $\mathit{points}(R) \cap \mathit{points}(R) = \{\}$.

However there may be *interface arrows* between the two graphlets.

Definition 5.2.

1. Let $P =_{df} \mathit{points}(R)$. The sets $\mathit{in}(R)$ of *input arrows* of R, $\mathit{out}(R)$ of *output arrows* of R, $\mathit{hid}(R)$ of *hidden arrows* of R and $\mathit{loose}(R)$ of *loose arrows* of R are characterised by
$$(a,b) \in \mathit{in}(R) \Leftrightarrow a \notin P \wedge b \in P\ ,\ (a,b) \in \mathit{out}(R) \Leftrightarrow a \in P \wedge b \notin P\ ,$$
$$(a,b) \in \mathit{hid}(R) \Leftrightarrow a \in P \wedge b \in P\ ,\ (a,b) \in \mathit{loose}(R) \Leftrightarrow a \notin P \wedge b \notin P\ .$$
With $A =_{df} \mathit{arrows}(R)$ these sets can be described algebraically as follows.
$$\mathit{in}(R) =_{df} \neg PAP\ , \qquad \mathit{out}(R) =_{df} PA\neg P\ ,$$
$$\mathit{hid}(R) =_{df} PAP\ , \qquad \mathit{loose}(R) =_{df} \neg PA\neg P\ ,$$
where $\neg P$ is the relative complement of P in the set I of all points, viz. $\neg P =_{df} I - P$. These sets are pairwise disjoint.
2. We call R *loose-free* if $\mathit{loose}(R) = \{\}$.
3. The *perimeter* of R is $\mathit{perimeter}(R) =_{df} \mathit{in}(R) + \mathit{out}(R)$.

Example 5.3.
In the graphlet from Ex. 3.2 we now draw the input arrows dashed and the output arrows dotted; the internal arrows remain solid. Moreover, we add some wiggly lose arrows. □



For loose-free R one has the decomposition
$$\mathit{arrows}(R) = \mathit{in}(R) + \mathit{out}(R) + \mathit{hid}(R) = \mathit{perimeter}(R) + \mathit{hid}(R)\ . \tag{4}$$
We will assume all graphlets considered to be loose-free unless stated otherwise.

We state some properties of interfaces.

Lemma 5.4. *Let R, S be point-disjoint, $P =_{df} \mathit{points}(R)$ and $Q =_{df} \mathit{points}(S)$.*

1. $\mathit{in}(R) \cap \mathit{in}(S) = \mathit{out}(R) \cap \mathit{out}(S) = \mathit{hid}(R) \cap \mathit{hid}(S) = \{\}$.
2. $\mathit{in}(R) \cap \mathit{hid}(S) = \{\} = \mathit{out}(R) \cap \mathit{hid}(S)$.
3. $\mathit{in}(R) \cap \mathit{out}(S) = Q(R \cap S)P$ *and* $\mathit{out}(R) \cap \mathit{in}(S) = P(R \cap S)Q$.
4. $R \cap S = (\mathit{in}(R) \cap \mathit{out}(S)) \cup (\mathit{in}(S) \cap \mathit{out}(R))$. *This means $R \cap S$ consists only of shared interface arrows; no internal arrows are involved.*

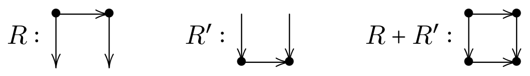
5. *$hid(R) \cup hid(S) \cup (R \cap S) \subseteq hid(R+S)$. This means that in connecting R and S using their interface arrows these become internal and hence are not connected to further graphlets.*
6. *$in(R)in(R) = \{\} = out(R)out(R)$.*

6 Disjoint Union

A number of our further composition operators on graphlets are variants of the union $R + S$ of point-disjoint graphlets R, S. This reflects that each event of a composite program is in the union of the events of its two operands, but no event occurs in more than one of them.

Definition 6.1. For doubly closed graphlets R, S we set $R + S =_{df} R \cup S$ provided $points(R) \cap points(S) = \{\}$. Then $points(R+S) = points(R) + points(S)$ and $arrows(R + S) = arrows(R) \cup arrows(S)$.

Example 6.2.



□

Lemma 6.3. *Let R and S be arrow-closed.*
1. *$in(R+S) = (in(R) - out(S)) + (in(S) - out(R))$.*
   $in(R+S) = (in(R) - out(S)) + (in(S) - out(R))$.
2. *$in(R+S) = (in(R) + in(S)) - (out(R) + out(S))$.*
   $out(R+S) = (out(R) + out(S)) - (in(R) + in(S))$.
3. *$perimeter(R+S) = perimeter(R) \,\Delta\, perimeter(S)$.*

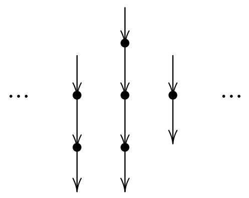
Part 1 says that the inputs of $R + S$ consists of all inputs of R that are not already 'satisfied' by the outputs of S, together with all the similarly unsatisfied inputs of S (a third equation for outputs is similar).

7 Lines

Henceforth we restrict attention to point-closed graphlets (cf. (2)).

A *fork* is a relation $\{(a,b),(a,c)\}$ where $b \neq c$. A relation R is *fork-free* if it contains no forks or equivalently, if it is a partial function, i.e., $R^{\smile}R \subseteq I$. *Join-freedom* is defined similarly, by interchanging R with $R^{\smile}$.

If R is both fork-free and join-free and $arrows(R)$ is acyclic then R is called *multilinear*. If $points(R)$ is finite then R can be pictured as a forest of trees without branches, i.e., only trunks which form straight lines.



Lemma 7.1. *Let R be acyclic. If additionally R is fork-free or join-free then R is intransitive.*

The informal reason is that a "transitive" arrow, like the one from x to z, that "bridges" the composition of two other arrows, like the ones from x to y and from y to z, necessarily induces a fork or a join at one of its ends (or at both).

$$\begin{tikzcd} x \arrow[r] \arrow[rr, bend left] & y \arrow[r] & z \end{tikzcd}$$

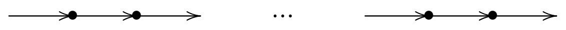
By Lm. 7.1 all multilinear relations are intransitive.

For relation R, as usual R^* is the reflexive-transitive closure of R, i.e., $R^* = R^+ \cup I$. Then the *R-interval* between $a, c \in \mathrm{EV}$ with $(a,a),(b,b) \in points(R)$ is

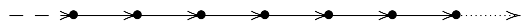
$$[a,c]_R =_{df} \{(b,b) \mid (b,b) \in points(R) \wedge (a,b) \in R^* \wedge (b,c) \in R^* c\}$$

A *line* is a multilinear relation R which is *bounded* and *connected*: this means that there are $a, b \in points(R)$ with $points(R) = [a,b]$. For similar relational characterisations of lines see [2,6]. In this case, R^* is a linear lax order on the points of R such that a is a least element and b is a greatest element.

Hence, if R^* is even a partial order, we set $least(R) =_{df} a$ and $greatest(R) =_{df} b$ when $[a,b] = points(R)$. Since for intransitive and acylic R the relation R^+ is a strict order, R^* then is indeed a partial order. Connectedness serves to exclude strange relations like



A line is called *open* if it has exactly one input and exactly one output arrow:



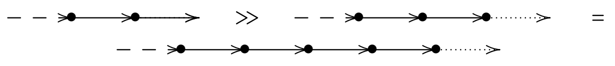
Open line R satisfies $least(R) = head(in(R))$ and $greatest(R) = tail(out(R))$; moreover, $points(R)$ is non-empty.

We define a concatenation operator $\gg$ on open lines R, S:

$$R \gg S =_{df} \begin{cases} R + S & \text{if } out(R) = in(S) \wedge out(S) \cap in(R) = \{\}\ , \\ \text{undefined} & \text{otherwise}\ . \end{cases}$$

Without the condition $out(R) = in(S)$ the result would have an extra input arrow and an extra output arrow and hence would not be a line. Without the condition $out(S) \cap in(R) = \{\}$ the result would have no inputs or outputs and would form a cyclic chain.

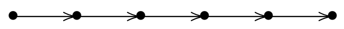
Example 7.2.



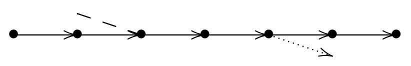
□

Theorem 7.3. *When defined, $R \gg S$ is again an open line with $points(R \gg S) = points(R) + points(S)$ as well as $in(R \gg S) = in(R)$ and $out(R \gg S) = out(S)$. Moreover, the operator $\gg$ is associative.*

We also use *closed* lines which are lines without input and output:



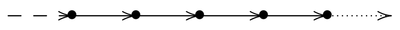
They represent, for instance, the behaviour of a single object by itself, from its creation to its deletion. When embedded in a larger system, ingoing and outgoing arrows may be present, reflecting communication with other objects.



8 Figures and Traversals

We now explore topological properties shared between graphlets, viewed as figures of our discrete geometry, and those of the more familiar Euclidean geometry.

A line is said to *pass through* a set A of arrows if it contains at least one arrow of A, and any such arrow is said to *lie on* the line. These definitions entail an analogue of the Jordan theorem of standard geometry. To formulate it we need an auxiliary notion. The set of *inner points* of an open line R is $\mathit{tails}(\mathit{arrows}(R)) \cap \mathit{heads}(\mathit{arrows}(R))$; this definition is reasonable by assumption (2) of closedness and ensures that $\mathit{tails}(\mathit{in}(R))$ and $\mathit{heads}(\mathit{out}(R))$ are not inner points of R:



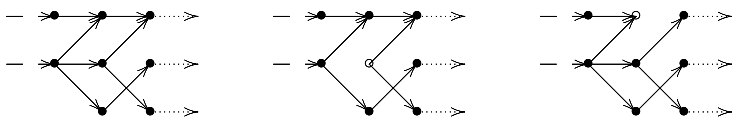
Only the inner points are depicted as bullets. The set of *inner points* of a graphlet R is the union of all inner points on open lines contained in R. All other points are called *outer points* of R. The *distance* between two inner points a, b of a line R is the number of R-arrows needed to get from a to b provided $a\,R^{+}\,b$; mathematically it is the least natural number n such that $a\,R^{n}\,b$.

Theorem 8.1. *Every open line from an outer point of R to an inner point of R passes through $\mathit{in}(R)$. Similarly, a line from an inner point to an outer point passes through $\mathit{out}(R)$.*

Proof. By induction on the distances between the points in question. □

Pursuing further our analogy with standard geometry, we will define a graphlet R to be *convex* if each of its arrows and points lies on a line which begins with an input arrow of R and ends with an output arrow. Such a line is called a *traversal* of R. A convex graphlet cannot have sources or sinks in its inner. We define a *figure* as a convex and cycle-free graphlet.

Example 8.2.
1. Every open line is a figure.
2. The empty graphlet (hereafter called 1) is a figure. Since it has no points, it is trivially convex and acyclic.
3. Of the following graphlets, the first is a figure whereas the others are not; we mark with circles the points at which convexity fails:



□

An *antichain* is a relation R such that $\mathit{heads}(R) \cap \mathit{tails}(R) = \{\}$. Therefore no two pairs in R connect. In particular, by (1), $\mathit{points}(R) = \{\}$. This means that all arrows of an antichain are loose and that the events of an antichain do not lie on lines passing through it. Hence antichains are not convex. Henceforth R and S stand for figures.

In conventional geometry, a convex figure may be split by any straight line from one point on its perimeter to another. Both sides of the split are obviously convex. Further, to specify the way in which the figure $R + S$ has been split into its components R and S, it suffices to specify just the way in which its perimeter has been split into a left side (taken from R) and a right side (from S).

Part II: Syntax and Semantics

9 Terms

Following [17], we take a syntax-oriented view of programs, namely in the form of abstract syntax trees. These correspond to the completely quadrangulated tracelets in [16]. We define program terms first by a context-free grammar and then impose restrictions to distinguish the valid ones for which a semantics can reasonably be defined. In a later section we define the semantics of a term as a program graph consisting of events in a computer linked by causality arrows.

We assume a set of *base terms* that are supposed to represent atomic commands or actions; it must contain the term $\{\}$ that represents the empty program graph. Furthermore we assume a set of binary composition operators, among them $+$, denoting some form of disjoint union. Then the set of *terms* is given by the context-free grammar

$$\langle\text{term}\rangle ::= \langle\text{base term}\rangle \mid (\langle\text{term}\rangle\ \langle\text{operator}\rangle\ \langle\text{term}\rangle)\ .$$

Usually we will omit the outermost pair of parentheses. The operators may denote partial functions of their arguments; hence the value of certain terms may be undefined. If defined, terms denote program graphs as defined in Sect. 3 below. Definedness is expressed by claims and assumptions, which also allow a general treatment of faults and violations.

10 Partialities: Assumptions and Claims

A *partial definition* of an operator $\circ$ is one which is qualified by an *assumption* that must be satisfied by the context of its use. If so, the application of $\circ$ is guaranteed to produce a value; otherwise the use of $\circ$ is *erroneous.* A definition may also be accompanied by a formula, called a *claim*, that describes some property of the result which is true of every valid use. If the result of the application of $\circ$ fails to satisfy the claim it is called *faulty* and otherwise *valid.* A faulty use is called a *counterexample* to the claim. It should be reported to the author of the claim, to assist in removal of the error. Assumptions and claims form also the basis of the familiar require/ensure and rely/guarantee approaches to program correctness.

An example is the definition of integer division div:

$$\begin{array}{lll} y \mathsf{\ div\ } z =_{df} x & \text{assuming} & y \text{ and } z \text{ are integers and } z \text{ is non-zero} \\ & \text{claiming} & x \text{ is an integer and } z\times x \le y < z\times(x+1) \end{array}$$

The above notions about terms are similar to ones used in “blame logics”. An assuming/claiming specification can be viewed as a contract between user and implementer. An erroneous term means a violation of that contract by the user (user error), while a faulty term hints at an implementation error (the user kept their part of the contract while the implementation did not). However, one has to beware of *unimplementable* specifications such as

$$\begin{array}{lll} x \mathsf{\ mysum\ } y =_{df} z, & \text{assuming} & x \text{ and } y \text{ are non-negative integers} \\ & \text{claiming} & z = x+y \text{ and } z < 0\ . \end{array}$$

Every term $x \mathsf{\ mysum\ } y$ is faulty, but neither user nor implementer can be blamed.

Conventional assertions and assumptions of a Boolean expression B are basic commands of many programming languages. Their partial definitions read

$$\begin{array}{lll} \mathsf{assert}\ B & =_{df} & \text{claiming} \quad B = \mathsf{true}\ , \\ \mathsf{assume}\ B & =_{df} & \text{assuming} \quad B = \mathsf{true}\ . \end{array}$$

One application of these constructs is in method declarations of a modern programming language: the assumed predicate is called a *precondition* and the claimed predicate is called a *postcondition.* The task of a debugging compiler is to produce counterexamples that will help the programmer to correct a violation, either in the calling program or in the method body.

We now propagate the notions of being erroneous, faulty and valid from operators inductively to composite terms. Moreover, given a set V of *values* we define a partial function *val* from the set of terms to V. In the case of graphlets V is the set of all binary relations over the set EV of events. We assume that for every non-erroneous term p a value $val(q) \in V$ can be defined. Then the *semantics* of term p is given by

$$[\![p]\!] =_{df} \begin{cases} val(p) & \text{if } p \text{ is valid} \\ \text{undefined otherwise} \end{cases}$$

In the above example of the operator myplus we have $val(x \mathsf{\,myplus\,} y) = val(x) + val(y)$ while $[\![x \mathsf{\,myplus\,} y]\!]$ is undefined.

For the mentioned inductive definitions we assume for each binary operator $\circ$ a binary *assumption predicate* $\circ$-*assu* and a unary *claim predicate* $\circ$-*claim*; the latter may involve the function *val*. These describe the “increment” in assumption/claim by the top level operator $\circ$ of a given term. Since they take terms and not values as arguments, they may access the components of these.

In the div example we have

$$\begin{array}{rl} \mathsf{div}\text{-}assu(y,z) & \Leftrightarrow_{df}\ val(y) \in \mathtt{int} \wedge val(z) \in \mathtt{int} \wedge val(z) \neq 0\ , \\ \mathsf{div}\text{-}claim(y \mathsf{\,div\,} z) & \Leftrightarrow_{df}\ w \in \mathtt{int} \wedge v\times w \le u < v\times(w+1)\ , \end{array}$$

where $u = val(y)$, $v = val(z)$ and $w = val(y \,\mathsf{div}\, z)$. Here it is essential that $y \,\mathsf{div}\, z$ is a *term* and not a value, because this still allows a unique extraction of the operand terms p and q.

We assume that the semantics of every operator $\circ$ is given by a binary partial function $\circ$-*func* on values. The requirements on *val* and $\circ$-*func* are detailed below in Req. 12.2.

To express the assumptions and claims for the base terms we additionally assume two unary predicates *atom_assu* and *atom_claim* on these. For instance, *atom_assu* or *atom_claim* might require a relation to be acyclic. We stipulate that $atom\_assu(\{\})$ and $atom\_claim(\{\})$ are true.

Now we inductively define *assu* and *claim* for composite terms. The idea is that a term with partially defined operators has as assumption/claim the conjunction of the assumptions/claims of all the operators that it contains, augmented by the assumption/claim of the operator itself.

Definition 10.1.

1. For atomic term p we set $assu(p) \Leftrightarrow_{df} atom\_assu(p)$. When $atom\_assu(p)$ holds, we set $claim(p) =_{df} atom\_claim(p)$.
2. For a composite term $p \circ q$ we set
$$\begin{aligned} assu(p \circ q) &\Leftrightarrow_{df} assu(p) \wedge assu(q) \wedge \circ\text{-}assu(p,q) \ , \\ claim(p \circ q) &\Leftrightarrow_{df} claim(p) \wedge claim(q) \wedge \circ\text{-}claim(p \circ q) \ . \end{aligned}$$
3. We stipulate that $\circ$-*assu* behaves well w.r.t. the empty relation, i.e., that $assu(p) \Rightarrow (\circ\text{-}assu(\{\},p) \wedge \circ\text{-}assu(p,\{\}))$. Then $\{\}$ composed with a non-erroneous term yields a non-erroneous term again.

11 Errors, Faults and Refinement

For the reader's benefit we repeat the definitions. Term p is *erroneous* if $\neg assu(p)$ holds. p is *faulty* if $assu(p) \wedge \neg claim(p)$ holds. A term p that is neither erroneous nor faulty is called *valid*; this is the case iff $assu(p) \wedge claim(p)$ holds. Clearly, the sets of erroneous, faulty and valid terms are pairwise disjoint.

We want to compare terms w.r.t. their degrees of definedness. A *lax order* (also known as a *preorder*) is a reflexive and transitive relation $\sqsubseteq$. It induces an equivalence relation $\sim_{\leq}$ by $p \sim_{\leq} q \Leftrightarrow_{df} p \leq q \wedge q \leq p$.

On valid terms p, q we assume a basic lax order $p \sqsubseteq q$. This might, for instance, be just equality of $val(p)$ and $val(q)$, as in the example of the div operator. But there are also instances where actually the syntactic structure of p and q plays a role. Below we use the internal flow of communication to distinguish sequential and parallel composition, using inclusion as the relation $\sqsubseteq$.

Based on this we now define a refinement relation between general terms.

Definition 11.1. We say that term p *refines* term q, in signs $p \leq q$, if the assumption of p implies that of q (which entails that if p is non-erroneous then so is q) and then if q is non-faulty then so is p and $p \sqsubseteq q$ holds. Formally,

$$p \leq q \Leftrightarrow (assu(p) \Rightarrow (assu(q) \wedge (claim(q) \Rightarrow (claim(p) \wedge p \sqsubseteq q))) \ . \qquad (5)$$

From this it is clear that we can proceed as follows to show $p \le q$.

- Assume $assu(p)$, because otherwise the refinement holds trivially.
- Show $assu(q)$ and then assume $claim(q)$, because otherwise the refinement holds trivially.
- Show $claim(p) \wedge p \sqsubseteq q$.

Theorem 11.2.

1. *$\le$ is a lax order.*
2. *All erroneous terms are least elements of the lax order $\le$ and hence equivalent w.r.t. $\sim_\le$. Hence we may define $\bot =_{df} \{p \mid \neg\, assu(p)\}$, i.e., as the equivalence class of all erroneous terms. By slight abuse of notation we also write $\bot$ instead of an arbitrary term in $\bot$.*
3. *All the faulty terms, i.e., the terms p with $assu(p) \wedge \neg claim(p)$, are equivalent and greatest elements of the lax order $\le$. Hence we may define $\top =_{df} \{p \mid assu(p) \wedge \neg claim(p)\}$, i.e., as the equivalence class of all faulty terms. By slight abuse of notation we write $\top$ instead of an arbitrary term in $\top$. By definition of $\sim$ we have $\top \le q \Leftrightarrow q \sim \top$.*
4. *For all operators $\circ$ and terms p we have $\bot \circ p \sim \bot \sim p \circ \bot$.*
5. *For all operators $\circ$ and terms p, if $\top \circ p$ is non-erroneous then $\top \circ p \sim \top$. Likewise, if $p \circ \top$ is non-erroneous then $p \circ \top \sim \top$.*
6. *On valid terms $\le$ coincides with $\sqsubseteq$.*
7. *The equivalence relation $\sim_\le$ satisfies*

$$\begin{aligned} p \sim_\le q \Leftrightarrow\ & (\neg assu(p) \wedge \neg assu(q)) \vee \\ & (assu(p) \wedge assu(q) \wedge \\ & \ ((\neg claim(p) \wedge \neg claim(q)) \vee (claim(p) \wedge claim(q) \wedge p \sim_\sqsubseteq q))) \ . \end{aligned}$$

By these properties, $\bot$ and $\top$ play the roles of the statements abort and miracle in the refinement calculi of [1,22,23,24]. In our present approach there is no need to introduce $\bot$ and $\top$ as extra syntactic constructs; they can be defined semantically as derived constructs.

To formulate the basic refinement relation $\sqsubseteq$ for the concrete case of relations we introduce a unary function *com* from relation terms to relations. It collects all the communications which pass within its parameter. To define *com* inductively, we require for each operator $\circ$ a unary function $\circ$-*com* on terms and set $com(p \circ q) =_{df} com(p) \cup com(q) \cup \circ\text{-}com(p \circ q)$ for valid terms p, q. Moreover, for base relation term p we set $com(p) =_{df} \{\}$ We require $\circ\text{-}com(\{\} \circ p) = \{\}$ and $\circ\text{-}com(p \circ \{\}) = \{\}$ for all p, since the empty set of events cannot create any communication. If nothing else is mentioned, $\circ\text{-}com(p \circ q)$ will be $\{\}$. With this we can now set

$$p \sqsubseteq q \Leftrightarrow_{df} com(q) \subseteq com(p) \ .$$

12 Modularity and Interchange

As mentioned in Sect. 6 many combination operators $\circ$ on graphlets are variants of the basic commutative and associative operator + of union of point-disjoint graphlets, differing just in their assumptions and claims. We assume for our general treatment of violations and refinement that there is an analogous operator $\oplus$ on terms whose $\oplus$-*func* is associative and commutative.

Definition 12.1. An operator $\circ$ is a *variant of* $\oplus$ if
- $\circ\text{-}\mathit{assu} \Rightarrow \oplus\text{-}\mathit{assu}$,
- $\mathit{assu}(p \circ q) \Rightarrow \mathit{val}(p \circ q) = \mathit{val}(p \oplus q)$.

Next, we detail the requirements on the *val* and $\circ$-*func* functions.

Requirement 12.2.
1. For atomic term p we assume $\mathit{val}(p)$ to be given.
2. For a composite term $p \circ q$ we set $\mathit{val}(p \circ q) =_{df} \circ\text{-}\mathit{func}(\mathit{val}(p), \mathit{val}(q))$ provided $\mathit{assu}(p \circ q)$.
3. The equivalence $\sim_{\leq}$ is a congruence for all operators $\circ$, i.e.,
$$p \sim_{\leq} q \Rightarrow p \circ r \sim_{\leq} q \circ r \ .$$
4. Generally we require for non-erroneous terms p and q that $\mathit{val}(p) = \mathit{val}(q) \Rightarrow (\mathit{claim}(p) \Leftrightarrow \mathit{claim}(q))$.

To prove further laws, we need an additional notion. A predicate $\circ$-*assu* is *modular* if it "distributes" through $\oplus$, i.e., if for all terms p, q, r with $\oplus\text{-}\mathit{assu}(p, q)$,

$$\begin{array}{l} \circ\text{-}\mathit{assu}(p \oplus q, r) \Leftrightarrow \circ\text{-}\mathit{assu}(p, r) \wedge \circ\text{-}\mathit{assu}(q, r) \quad \text{and} \\ \circ\text{-}\mathit{assu}(r, p \oplus q) \Leftrightarrow \circ\text{-}\mathit{assu}(r, p) \wedge \circ\text{-}\mathit{assu}(r, q) \ . \end{array}$$

Likewise, a function $\circ$-*com* is *submodular* if it subdistributes through $\oplus$, i.e., if for all terms p, q, r with $\circ\text{-}\mathit{assu}(p, q)$,

$$\begin{array}{l} \circ\text{-}\mathit{com}((p \oplus q) \circ r) \subseteq \circ\text{-}\mathit{com}(p \circ r) \oplus \circ\text{-}\mathit{com}(q \circ r) \quad \text{and} \\ \circ\text{-}\mathit{com}(r \circ (p \oplus q)) \subseteq \circ\text{-}\mathit{com}(r \circ p) \oplus \circ\text{-}\mathit{com}(r \circ q) \ . \end{array}$$

If the inclusions strengthen to equations then $\circ$-*com* is *modular*. If $\circ\text{-}\mathit{com}(p, q) = \{\}$ for all p, q then $\circ$-*com* is trivially submodular and modular.

Modularity was called bilinearity in [12], but this would conflict with our notion of multilinearity.

Lemma 12.3. *The conjunction of modular predicates is modular.*

Theorem 12.4. *Let $\circ_1$-assu and $\circ_2$-assu be modular predicates and $\circ_1$-com and $\circ_2$-com functions such that for all p, q with $\oplus\text{-}\mathit{assu}(p, q)$ satisfy $\circ_1\text{-}\mathit{assu}(p, q) \Rightarrow \circ_2\text{-}\mathit{assu}(p, q)$ and $\circ_1\text{-}\mathit{com}(p, q) \subseteq \circ_2\text{-}\mathit{com}(p, q)$. Then $\circ_1$ and $\circ_2$ satisfy the* small interchange laws

$$(p \circ_2 q) \circ_1 r \leq p \circ_2 (q \circ_1 r) \ , \qquad p \circ_1 (q \circ_2 r) \leq (p \circ_1 q) \circ_2 r \ .$$

Corollary 12.5. *If $\circ$-assu is modular then $\circ$ is associative up to $\sim$.*

Theorem 12.6. *Consider operators $\circ_1$ and $\circ_2$ such that*
- *$\circ_1$-assu, $\circ_1$-assu and $\circ_2$-com are modular and $\circ_1$-com is submodular,*
- *all p, q with $\oplus\text{-}\mathit{assu}(p, q)$ satisfy $(\circ_1\text{-}\mathit{assu}(p, q) \Rightarrow \circ_2\text{-}\mathit{assu}(p, q))$ as well as $\circ_1\text{-}\mathit{com}(p, q) \subseteq \circ_2\text{-}\mathit{com}(p, q)$,*

– *$\circ_2$-assu is symmetric, i.e., $\circ_2$-assu$(p,q) \Leftrightarrow \circ_2$-assu$(q,p)$, and $\circ_2$-com is commutative, i.e., $\circ_2$-com$(p,q) = \circ_2$-com(q,p).*

Then $\circ_1$ and $\circ_2$ satisfy the full interchange law

$$(p \circ_2 q) \circ_1 (r \circ_2 s) \leq (p \circ_1 r) \circ_2 (q \circ_1 s) \ .$$

13 Concrete Composition Operators

13.1 Union

To allow flexible generation we also admit a general binary union operator $\circledcup$ on graphlet terms and give it a formal partial definition in the sense of Def. 10.1. We set, for terms p,q,

$$\begin{array}{lll} \circledcup\text{-}\mathit{assu}(p,q) & \Leftrightarrow_{df} & \mathsf{true} \\ \circledcup\text{-}\mathit{claim}(p) & \Leftrightarrow_{df} & \mathsf{true}\ , \\ \circledcup\text{-}\mathit{com}(p,q) & \Leftrightarrow_{df} & \mathit{val}(p) \cup \mathit{val}(q)\ , \\ \circledcup\text{-}\mathit{func}(R,S) & =_{df} & R \cup S\ . \end{array}$$

This definition may appear cyclic, since $\mathit{val}(p)$ and $\mathit{val}(q)$ are used although it is not clear that p and q are non-erroneous. However, since $\mathit{assu}(p \circledcup q) \Leftrightarrow \mathit{assu}(p) \wedge \mathit{assu}(q) \wedge \circledcup\text{-}\mathit{assu}(p,q)$, the definition of $\cup\text{-}\mathit{assu}(p,q)$ occurs in a context where $\mathit{assu}(p)$ and $\mathit{assu}(q)$ hold.

Example 13.1. We assume that there are atomic terms denoting the events. To save notation we simply use the event names for these. Let a,b,c be event terms let ; denote sequential composition (details are given in Sect. 13.3). Then the N shape of Rem. 3.4 is, for instance, denoted by the term

$$((a\,;c) \circledcup (a\,;d)) \circledcup (b\,;d)\ .$$

□

13.2 Disjoint Union

In Sect. 5 we have defined graphlets to be *point-disjoint* if their intersection contains no points. In that case their union was denoted by $R + S$.

To give a formal partial definition of the operator $\oplus$ on graphlet terms in the sense of Def. 10.1 we set, for terms p,q,

$$\begin{array}{lll} \oplus\text{-}\mathit{assu}(p,q) & \Leftrightarrow_{df} & \mathit{points}(\mathit{val}(p)) \cap \mathit{points}(\mathit{val}(q)) = \{\} \\ \oplus\text{-}\mathit{claim}(p) & \Leftrightarrow_{df} & \mathsf{true}\ , \\ \oplus\text{-}\mathit{com}(p,q) & \Leftrightarrow_{df} & \mathit{val}(p) \cap \mathit{val}(q)\ , \\ \circ\text{-}\mathit{func}(R,S) & =_{df} & R + S\ . \end{array}$$

By Def. 12.1 every operator $\circ$ that that is a variant of $\oplus$ on graphlets satisfies

$$\circ\text{-}\mathit{func}(R,S) = R + S\ .$$

By this and Def. 10.1.3, $\{\}$ is the unit of every such $\circ$.

Note that the analogue of the term in Ex. 13.1 with $\oplus$ instead of $\circledcup$ is erroneous, since the graphlets formed using ; are not point-disjoint.

13.3 Sequential Composition

For example, we define the operator ; of *sequential composition.* This is designed to ensure that its first argument can be executed in its entirety before starting the second argument. Formally, we use the predicates

$$\begin{array}{lll} ;\text{-}\mathit{assu}(p,q) & \Leftrightarrow_{df} & \oplus\text{-}\mathit{assu}(p,q) \wedge \mathit{in}(\mathit{val}(p)) \cap \mathit{out}(\mathit{val}(p)) = \{\} \ , \\ ;\text{-}\mathit{claim}(p) & \Leftrightarrow_{df} & \mathsf{true} \ , \\ ;\text{-}\mathit{com}(p) & \Leftrightarrow_{df} & \mathit{out}(\mathit{val}(p)) \cap \mathit{in}(\mathit{val}(p)) \ . \end{array}$$

The assumption $;\text{-}\mathit{assu}(p,q)$ reflects that if there is a causal dependence arrow pointing from p to q, then it is physically impossible to complete execution of p before beginning q.

Theorem 13.2. *Over loose-free graphlets the predicate* ;-*assu is modular, and hence for such graphlets* ; *is associative up to equivalence.*

The proof just uses distributivity of intersection over +.

13.4 Disjoint Concurrent Composition

Our first variant $|||$ of concurrent composition of graphlets is designed to ensure that that its two arguments can be executed on separate computers, with no communication between them. Its assumption and claim are

$$\begin{array}{lll} |||\text{-}\mathit{assu}(p,q) & \Leftrightarrow_{df} & \oplus\text{-}\mathit{assu}(p,q) \wedge \mathit{in}(\mathit{val}(p)) \cap \mathit{out}(\mathit{val}(q)) = \{\} = \\ & & \mathit{in}(\mathit{val}(q)) \cap \mathit{out}(\mathit{val}(p)) \ , \\ |||\text{-}\mathit{claim}(p) & \Leftrightarrow_{df} & \mathsf{true} \ , \\ |||\text{-}\mathit{com}(p) & \Leftrightarrow_{df} & \{\} \ . \end{array}$$

Theorem 13.3. *Up to equivalence,* ; *and* $|||$ *have the same unit* $\{\}$.

The definition implies $|||\text{-}\mathit{assu}(R,S) \Leftrightarrow ;\text{-}\mathit{assu}(R,S) \wedge ;\text{-}\mathit{assu}(S,R)$ and hence $|||\text{-}\mathit{assu}$ is modular as a conjunction of modular predicates, which by Cor. 12.5 implies the following.

Theorem 13.4. $|||$ *commutes and is associative, up to equivalence.*

Lemma 13.5.
1. $\mathit{in}(R_1 \,|||\, R_2) = \mathit{in}(R_1) + \mathit{in}(R_2)$.
2. $\mathit{out}(R_1 \,|||\, R_2) = \mathit{out}(R_1) + \mathit{out}(R_2)$.
3. $\mathit{hid}(R_1 \,|||\, R_2) = \mathit{hid}(R_1) + \mathit{hid}(R_2)$.

An alternative form $|$ of concurrent composition permits internal communication between its operands, but weakens the claim condition to require only that there is no cyclic chain of arrows that crosses between them.

$$\begin{array}{lll} |\text{-}\mathit{assu}(p,q) & \Leftrightarrow_{df} & \oplus\text{-}\mathit{assu}(p,q) \wedge \mathit{cyclefree}(\mathit{val}(p)) \wedge \mathit{cyclefree}(\mathit{val}(q)) \ , \\ |\text{-}\mathit{claim}(p) & \Leftrightarrow_{df} & \mathit{cyclefree}(\mathit{val}(p)) \ , \\ |\text{-}\mathit{com}(p,q) & \Leftrightarrow_{df} & \oplus\text{-}\mathit{com}(p,q) \end{array}$$

13.5 Concatenation

The concatenation operator $p \gg q$ can be extended from lines to more general figures by weakening the definition: the outputs of $val(p)$ do not have to be the same as the inputs of $val(p)$.

More precisely, call lines $R \subseteq val(p)$ and $S \subseteq val(q)$ *concatenable* if $R \gg S$ is defined. Then $\gg$-$func(p,q)$ consists of all concatenations of concatenable lines from $val(p)$ and $val(q)$ together with all other lines from $val(p)$ and $val(q)$. The formal definition is as follows.

$$\begin{aligned}
\gg\text{-}assu(p,q) &\Leftrightarrow_{df} \oplus\text{-}assu(p,q) \wedge \wedge\, cyclefree(val(p)) \wedge cyclefree(val(q)) \wedge \\
&\phantom{\Leftrightarrow_{df}}\ in(val(p)) \cap out(val(p)) = \{\}\ , \\
\gg\text{-}claim(p) &\Leftrightarrow_{df} cyclefree(val(p))\ , \\
\gg\text{-}com(p) &\Leftrightarrow_{df} out(val(p)) \cap in(val(p))\ .
\end{aligned}$$

Acyclicity of the result is assured by acyclicity of the operands and by continuing to insist that that the inputs of $val(p)$ are disjoint from the outputs of $val(q)$; thus no cyclicity can arise in their concatenation.

Since $\gg$-$assu$ coincides with ;-$assu$, the same proof technique can be used to show that it is modular.

Theorem 13.6. $\gg$ *on figures obeys the same laws as those in Th. 7.3 for lines.*

13.6 Concurrent Composition

A further weakening of the definition of sequential composition leads to a *concurrent composition operator* $p \,||\, q$. The weakening removes the restriction that the inputs of p must be disjoint from the outputs of q. The result can therefore contain a cycle, even when both $val(p)$ and $val(q)$ are acyclic. And of course, if the operands are not point-disjoint, the result will be undefined.

We define $||$ formally by using the predicate $||$-$assu(p,q) \Leftrightarrow_{df} +$-$assu(p,q)$ and by $||$-$com(p,q) =_{df} val(p) \cap val(q)$, i.e.,

$$com(p \,||\, q) = com(p) \cup com(q) \cup (val(p) \cap val(q))\ .$$

By distributivity of intersection over disjoint union $||$-com is modular.

This definition implies that $||$ and $\oplus$ coincide.

Theorem 13.7. *If $p||q$ is valid and $val(p)$ and $val(q)$ are figures then $val(p||q)$ is a figure.*

Proof. (Sketch) We show that every traversal of p or q can be extended to a traversal of $p \,||\, q$. Take a traversal T of p. If its last arrow does not pass through the perimeter shared by p and q then it is an output arrow of $p||q$, so that T is a traversal of $p||q$ as well. Otherwise its last arrow leads into q and we can chain it with a traversal of q starting with that arrow, which again yields a traversal of $p \,||\, q$. Since p and q are figures, all their points and arrows are contained in their traversals, so that the above process covers all points and arrows of $p \,||\, q$. □

By the properties mentioned in Sect. 11 the following theorem is immediate.

Theorem 13.8. $||$ *is associative, commutative and isotone, up to* $\sim$*. Moreover,* $\gg$ *and* $||$ *satisfy the full interchange law and hence the small interchange laws, as do* $|||$ *and* $||$ *as well as* ; *and* $||$.

14 Programs

14.1 From Graphlets to Programs: Lifting

So far we have dealt with single graphlets. A *program* is identified by and with the set of all terms denoting possible graphlets of its execution. This section explains how operators on graphlets can be lifted to sets of graphlets in such a way that the laws proved for the graphlet operators are preserved.

14.2 Elementwise Lifting

As in [15] we do not consider arbitrary sets of graphlet terms. A set G of graphlet terms is a *program* if it is *downward closed* w.r.t. the lax refinement order $\leq$, i.e., if $p \in G$ and $p' \leq p$ imply $p' \in G$ as well. Downward closure codifies our intention that any program that can validly be executed concurrently can also be validly executed more sequentially.

If $\circ$ is a binary, possibly partial, operator on graphlet terms then its *elementwise lifting* to programs G, G' is defined as the downward closure of the set of all defined compositions between G and G', i.e., the set of all graphlet terms q such there are $p \in G$ and $p' \in G'$ with valid term $p \circ p'$ and $q \leq p \circ p'$.

Since we use inequational laws, we need to define a refinement relation between programs if we want to lift laws to programs. There are several ways to extend a lax order like $\leq$ to sets. We choose the following definition: $G \leq G'$ holds iff every term in G is below some term in G'. For downward closed sets (and hence programs) $\leq$ coincides with inclusion $\subseteq$. Hence we can define non-deterministic choice as set union in our program algebra.

Let p, p' be graphlet terms. A sufficient condition for lifting a law $p \leq p'$ law from graphlets to programs is *linearity*, viz. that every variable occurs at most once in p, p' and that all variables in p also occur in p'. Examples are the frame and exchange laws. For equations a sufficient condition is *bilinearity*, meaning that both inequations that constitute an equation are linear. Examples are associativity, commutativity and neutrality; a counterexample is distributivity. The main result is as follows (cf. [15]).

Theorem 14.1. *If a linear law $p \leq p'$ holds for graphlet terms then it also holds when all variables in p, p' are interpreted as variables for programs and the operators are interpreted as the elementwise liftings of the corresponding graphlet operators.*

14.3 Errors, Recursion and Iteration

There are further useful consequences of our definition of programs. The set $\mathcal{G}$ of all programs forms a complete lattice w.r.t. the inclusion ordering; it has been called the *Hoare power domain* (e.g. [26,18,4]). The least element of $\mathcal{G}$ is the empty program $\{\}$ which can also serve as an error element, modelling a completely faulty module without any sensible graphlet. A more detailed, elementwise, error handling is already contained in the definition of the elementwise lifting of operators: all erroneous, undefined combinations of graphlets are ruled out from the combination of the containing programs. The greatest element of $\mathcal{G}$ is the program consisting of all graphlet terms. Infimum and supremum in $\mathcal{G}$ coincide with intersection and union, since downward closed sets are also closed under these operations. Thus we can define (unbounded) choice between a set $\mathcal{Q} \subseteq \mathcal{G}$ of programs as $[]\, \mathcal{Q} =_{df} \bigcup \mathcal{Q}$ with binary choice as the special case $G \,[]\, G' =_{df} G \cup G'$. This means that an implementation can make an arbitrary choice between non-deterministic variants allowed by the program under execution, giving our intended interpretation of non-determinism a demonic flavour.

The lifted versions of isotone graphlet operators are isotone again (see [15]), but even distribute through arbitrary choices between programs.

Isotony of the lifted operators, together with completeness of the lattice of programs and the Tarski-Knaster fixed point theorem, guarantees solutions of recursion equations. More precisely, let $f : \mathcal{G} \to \mathcal{G}$ be an isotone function. Then f has a least fixed point μf and a greatest fixed point νf, given by

$$\mu f = \bigcap \{G \mid f(G) \subseteq G\} \ , \qquad \nu f = \bigcup \{G \mid G \subseteq f(G)\} \ .$$

With our operator ; this can be used to define the Kleene star (see e.g. [5]), i.e., unbounded finite sequential iteration, of a program G as $G^* =_{df} \mu f_P$, where $f_P(X) =_{df} \mathsf{skip} \,[]\, (G\,;X)$ and $\mathsf{skip} =_{df} \{\Box\}$ is the idle program. Since by the above remark f_P distributes through arbitrary choices, it is even continuous and Kleene's fixed point theorem tells us that G^* has the iterative representation $G^* = \bigcup\{G^i \mid i \in \mathbb{N}\}$ with $G^0 =_{df} \mathsf{skip}$ and $G^{i+1} =_{df} G\,;G^i$. Infinite iteration G^ω can be defined as the greatest fixed point νg_P where $g_P(X) =_{df} G\,;X$.

Along the same lines, unbounded finite and infinite concurrent iteration of a program can be defined. For further forms of iteration we refer to [15].

We conclude this section with a brief description how pre-post-condition semantics can be integrated into our approach. As in [12] one can define, for programs G, G' and Q, the Hoare triple

$$G \ \{Q\}\ G' \Leftrightarrow_{df} G\,;Q \subseteq G' \ .$$

It expresses that, after any graphlet in "pre-history" G, execution of Q is guaranteed to yield an overall graphlet in G'. This implies the standard properties of Hoare logic and separation logic; for further details we refer to [12,13].

15 Outlook: Concrete Programming Concepts

We now briefly describe how our geometric notions can be used to model concepts from current programming languages.

The points in a diagram represent events in a computer, such as assignments or communications.

Using lines we can define coordinate systems and how points are placed in them. We use two coordinate directions: vertical for succession in time (flowing downwards) and horizontal for distribution in space. To avoid clutter arrows are drawn as simple line segments. As labels we use, for instance, variable/value associations, channel names with input/output actions etc.

Using diagrams of this kind, a number of characteristic programming ideas and intuitions can be conveyed, namely

- assignments to local memory,
- transactions (atomic events, Petri net transitions),
- allocation and disposal of objects,
- variables (in stack or heap), input and output ports, threads,
- distribution and concurrency,
- buffered communications on reliable channels,
- synchronous communications and fences,
- shared memory with resource ownership,
- release and acquisition of resources between threads,
- message-sequence charts.

Example 15.1. We consider the following simple program. The two parallel vertical lines in the centre of the program are a parenthesis-avoiding form of the operator || from Sect. 13.6. The program uses objects, like the variables t,u,x,y, channels c,d and memory locations c[84],d[37] within c,d.

```
{ new x ;      ||  { y := y + 1 ;
  c[84]?x ;    ||    acq x ;
  rel x ; }    ||    d[37]!(y + x) ;
               ||    disp x ; }
```

The left parallel branch of the program allocates x as a new variable and claims ownership of it, subsequently executes a read command ? putting the value from channel location c[84] into x and then releases ownership of x. The right parallel branch first increments the global variable y, then acquires ownership of x, executes a write command ! putting the value y+x into channel location d[37] and finally deallocates x.

A diagram representing the case of an initial value of 3 for y is shown in Fig. 15. The solid vertically oriented lines depict threads exhibiting the sequential behaviours of single objects. Points connected by horizontal dashed lines stand for events that occur simultaneously, at the same instant of time. The sloping arrow in the diagram centre stands for a transfer of ownership of the variable x from one object thread to the other. The other two sloping arrows represent the input and output channels. □

More details are given in the companion paper [21].

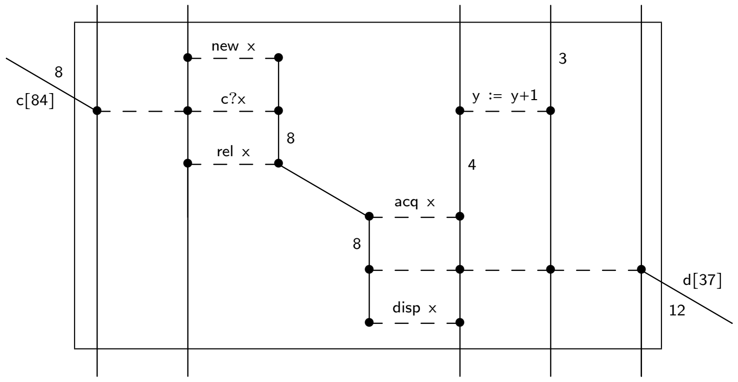


Fig. 1. Graphical representation of the sample program

Acknowledgment The authors would like to thank the Isaac Newton Institute for Mathematical Sciences, Cambridge, for support and hospitality during the programme *Verified Software* where work on this paper was undertaken, partially funded by EPSRC grant no EP/R014604/1. Helpful comments were provided by Roland Glück and Phil Wadler.

References

1. R. Back: A calculus of refinements for program derivations. Acta Inf. 25(6), 593–624 (1988)
2. R. Berghammer, H. Furusawa, W. Guttmann, P. Höfner: Relational characterisations of paths. CoRR abs/1801.04026 (2018), http://arxiv.org/abs/1801.04026
3. C. Brink: Power structures. Alg. Univ. 30(2), 177–216 (1993)
4. C. Brink, I. Rewitzky: A paradigm for program semantics: power structures and duality. CSLI Publications (2001)
5. J. Conway: Regular Algebra and Finite Machines. Chapman and Hall (1971)
6. H.-H. Dang, B. Möller: Transitive separation logic. In: W. Kahl, T. G. Griffin (eds.): Relational and Algebraic Methods in Computer Science. LNCS 7560. Springer 2012, 1–16. Extended version: H.-H. Dang, B. Möller: Extended transitive separation Logic. J. Log. Algebr. Meth. Program. 84(3): 303–325 (2015)
7. U. Fahrenberg, C. Johansen, G. Struth, R. Bahadur Thapa: Generating posets beyond N. In U. Fahrenberg, P. Jipsen, M. Winter (eds.): Relational and Algebraic Methods in Computer Science LNCS 12062. Springer 2020, 82–99
8. N. Gautam: The validity of equations of complex algebras. Arch. Math. Logik Grundl. Mat. 443, 117–124 (1957)
9. J. Gischer: The equational theory of pomsets. Theo. Comp. Sci. 61, 199–224 (1988)
10. R. Goldblatt: Varieties of complex algebras. Ann. Pure Appl. Logic 44, 173–242 (1989)
11. Grätzer, G., Whitney, S.: Infinitary varieties of structures closed under the formation of complex structures. Coll. Math. 48, 1–5 (1984)
12. T. Hoare, B. Möller, G. Struth, I. Wehrman: Concurrent Kleene Algebra and its foundations. J. Log. Algebr. Program. 80(6): 266–296 (2011)

13. T. Hoare, A. Hussain, B. Möller, P. O'Hearn, R. Petersen, G. Struth, G.: On locality and the exchange law for concurrent processes. In: J. Katoen, B., König, B. (eds.) CONCUR — Concurrency Theory. LNCS 6901. Springer 2011, 250–264
14. T. Hoare, S. van Staden, B. Möller, G. Struth, J. Villard, H. Zhu, P. O'Hearn. Developments in Concurrent Kleene Algebra. In P. Höfner, P. Jipsen, W. Kahl, M. E. Müller (eds.): Relational and Algebraic Methods in Computer Science. LNCS 8428. Springer 2014, 1–18
15. Hoare, T., van Staden, S., Möller, B., Struth, G., Zhu, H.: Developments in Concurrent Kleene Algebra. J. Log. Algebr. Meth. Program. 85(4), 617–636 (2016)
16. B. Möller, T. Hoare, M.E. Müller, G. Struth: A discrete geometric model of concurrent program execution. In J. Bowen, H. Zhu (eds.): Unifying Theories of Programming. LNCS 10134. Springer 2016, 1–25
17. T. Hoare, G. Struth, J. Woodcock: A calculus of space, time, and causality: its algebra, geometry, logic. In P. Ribeiro, A. Sampaio (eds.): Unifying Theories of Programming. LNCS 11885. Springer 2019, 3–21
18. M. Main: A powerdomain primer. Tech. Rep. CU-CS-375-87 (1987). Paper 360, Univ. Colorado at Boulder, Dept of Computer Science (1987), `http://scholar.colorado.edu/csci_techreports/360`
19. `W. McCune: Prover9 and Mace4. http://www.cs. unm.edu/~mccune/mace4/`
20. B. Möller, T. Hoare: Exploring an interface model for CKA. In R. Hinze, J. Voigtländer (eds.): Mathematics of Program Construction. LNCS 9129. Springer 2015, 1–29
21. B. Möller, T. Hoare, Z. Hou, J. Song Dong: Geometric Theory for Program Testing. Preprint 2022. DOI: 10.48550/arXiv.2206.02083
22. C. Morgan: The specification statement. ACM TOPLAS 10(3), 403–419 (1988)
23. J. Morris: A theoretical basis for stepwise refinement and the programming calculus. Sci. of Comp. Prog. 9, 287–306 (1987)
24. G. Nelson: A generalization of Dijkstra's calculus. ACM Trans. Prog Lang. Syst. 11:4, 517–561 (1989)
25. G. Schmidt, T. Ströhlein: Relations and Graphs: Discrete mathematics for computer scientists. EATCS Mon. Theo. Comp. Sci. Springer 1993
26. G. Winskel: On powerdomains and modality. Theo. Comp. Sci. 36, 127–137 (1985)

16 Appendix A: Further Relational Notions

16.1 Converse

The operator $\breve{}$ of *converse* mirrors all pairs of a relation:

$$R\breve{} = \{(b,a) \mid (a,b) \in R\} \ ,$$

For instance, $<\breve{} = >$.

We recall the fundamental rules of Dedekind and Schröder about binary relations R, S, T (e.g. [25]):

$$RS \cap T \subseteq R(S \cap R\breve{}T) \ , \qquad \text{(Dedekind I)}$$

$$RS \cap T \subseteq (R \cap TS\breve{})S \ , \qquad \text{(Dedekind II)}$$

$$RS \subseteq T \Leftrightarrow R\breve{}\overline{T} \subseteq \overline{S} \Leftrightarrow \overline{T}S\breve{} \subseteq \overline{R} \ . \qquad \text{(Schröder)}$$

16.2 Acyclicity and Orders

A relation R is called *acyclic* if it does not admit a proper path from any point to itself, i.e., if $R^+ \cap I = \{\}$, where R^+ is the transitive closure of R. Hence R is acyclic iff R^+ is irreflexive. Since $R \subseteq R^+$, every acyclic relation is irreflexive. If a relation is cyclic, then every other relation which contains it is also cyclic.

Frequently an order-theoretic view of relations is useful. A relation R is called a *strict-order* if it is *irreflexive* and *transitive*, i.e., if $R \cap I = \{\}$ and $RR \subseteq R$. It is called *asymmetric* if it does not relate elements mutually, i.e., if $R \cap R^{\smile} = \{\}$.

Lemma 16.1.
1. *Every asymmetric relation is irreflexive.*
2. *Every strict-order is asymmetric.*

Proof.
1. It is known that every subrelation of I is symmetric. Hence for arbitrary asymmetric relation R
$$R \cap I = (R \cap I) \cap (R \cap I)^{\smile} \subseteq R \cap R^{\smile} = \{\} .$$
2. By neutrality of I, (Dedekind II) with $(R^{\smile})^{\smile} = R$, transitivity if R, irreflexivity of R and strictness of ; :
$$R \cap R^{\smile} = R \cap I \,;R^{\smile} \subseteq (R\,; R \cap I)\,;R^{\smile} \subseteq (R \cap I)\,;R^{\smile} = \{\}\,;R^{\smile} = \{\} .$$

□

From the definitions and Lm. 16.1 the following result is immediate.

Lemma 16.2. *The following statements are equivalent.*
1. *R is acyclic.*
2. *R^+ is a strict-order.*
3. *R^+ is asymmetric.*

For the next result we need an auxiliary notion. A relation R is *intransitive* if $R \cap R^+R^+ = \{\}$. This means that no arrow of R can be "inferred" as a composition of at least two arrows of R.

Lemma 16.3. *An intransitive relation is irreflexive.*

This is immediate from Lm. 16.2.

If R is intransitive and R^+ is a strict-order (i.e., if R is also acyclic) then R is called a *Hasse diagram* for R^+ and represents the immediate successor relation for R^+. It is well known that a dense relation R (defined by $R \subseteq RR$, such as $<$ on the rationals or the reals) does not have a Hasse diagram.

Finally, we repeat some well-known properties of reflexive-transitive and transitive closure.

Lemma 16.4. *Consider relations R, S.*
1. *If $SR = \{\}$ then $SR^* = S$ and $SR^+ = \{\}$.*
2. *Therefore $(R \cup S)^* = R^*S^*$ and $(R \cup S)^+ = R^+ \cup R^+S^+ \cup S^+ = R^+ \cup R^*S^+ = R^+S^* \cup S^+$.*

3. If also $RS = \{\}$ *then* $(R \cup S)^* = R^* \cup S^*$ *and* $(R \cup S)^+ = R^+ \cup S^+$.

Proof.

1. By a fixpoint equation for *, distributivity, neutrality of I with the assumption and strictness with neutrality of $\{\}$:
$$SR^* = S(I \cup RR^*) = SI \cup SRR^* = SI \cup \{\}R^* = S \ .$$
2. First, by star of union and Part 1:
$$(R \cup S)^* = R^*(SR^*)^* = R^*S^* \ .$$
Second, by the definition of plus, the first claim, distributivity, Part 1 and the definition of plus:
$$\begin{aligned}(R \cup S)^+ &= (R \cup S)(R \cup S)^* = (R \cup S)R^*S^* = RR^*S^* \ \cup \ SR^*S^* \\ &= RR^*S^* \ \cup \ SS^* = R^+S^* \ \cup \ S^+ \ .\end{aligned}$$
The remaining equations are shown by straightforward regular algebra.
3. First, by Part 2, fixpoint equations for *, distributivity with neutrality of I, assumption, strictness with neutrality of $\{\}$ and Boolean algebra with fixpoint equations for *:
$$\begin{aligned}(R \cup S)^* &= R^*S^* = (I \cup R^*R)(I \cup SS^*) = I \ \cup \ SS^* \ \cup \ R^*R \ \cup \ R^*RSS^* \\ &= I \ \cup \ SS^* \ \cup \ R^*R \ \cup \ R^*\{\}S^* = I \ \cup \ SS^* \ \cup \ R^*R = S^* \cup R^* \ .\end{aligned}$$
Second, by definition of plus, the first claim, distributivity, Part 1 and definition of plus with regular algebra:
$$\begin{aligned}(R \cup S)^+ &= (R \cup S)(R \cup S)^* = (R \cup S)(R^* \cup S^*) \\ &= RR^* \cup RS^* \cup SR^* \cup SS^* = RR^* \cup R \cup S \cup SS^* = R^+ \cup S^+ \ .\end{aligned}$$

□

17 Appendix B: Proofs for the Main Text

We use the following well known facts about point sets $P, Q \subseteq I$:
$$PQ = P \cap Q = QP \ , \qquad P = P^{\smile} \ . \tag{6}$$
Moreover, We frequently use the following interplay between restriction and intersection, where P is a point sets and A, B are arrow sets:
$$PA \cap B = A \cap PB = P(A \cap B) \text{ and } AB \cap P = A \cap BP = (A \cap B)P \ . \tag{7}$$
Consequently, if $PQ = \{\}$ for point sets P, Q then, by strictness of relational composition,
$$PA \cap QB = \{\} = AP \cap BQ \ .$$

Since *tails* and *heads* coincide with the domain and range functions on relations, we have
$$\begin{array}{ll} heads(R^{\smile}) = tails(R) \ , & tails(R^{\smile}) = heads(R) \ , \\ heads(RS) \subseteq heads(S) \ , & tails(RS) \subseteq tails(R) \ , \\ heads(R^+) = heads(R) \ , & tails(R^+) = tails(R) \ . \end{array} \tag{8}$$

Proof of Lm. 5.4. Consider point-disjoint R, S and set $P =_{df} \mathit{points}(R)$, $Q =_{df} \mathit{points}(S)$. By relational and Boolean algebra,

$$PQ = \{\} \Leftrightarrow P \cap Q = \{\} \Leftrightarrow P \leq \neg Q \Leftrightarrow P \cap \neg Q = P \ .$$

1. As a sample,

$$\begin{array}{ll} & \mathit{in}(R) \cap \mathit{in}(S) \\ = & \neg PRP \cap \neg QSQ & \{\!\![\text{ the definitions }]\!\!\} \\ = & \neg PRP \cap \neg QSQP & \{\!\![\text{ restriction (7) }]\!\!\} \\ = & \neg PRP \cap \neg QS\{\} & \{\!\![\text{ point-disjointness of } R, S\ ]\!\!\} \\ = & \{\} \ . & \{\!\![\text{ strictness of composition and intersection }]\!\!\} \end{array}$$

   The other properties are proved analogously.

2. Analogous to Part 1.

3. $$\begin{array}{lll} & R \cap S & \\ = & (\mathit{in}(R) + \mathit{out}(R) + \mathit{hid}(R)) \cap & \{\!\![\ (4)\ ]\!\!\} \\ & (\mathit{in}(S) + \mathit{out}(S) + \mathit{hid}(S)) & \\ = & (\mathit{in}(R) \cap \mathit{in}(S)) + (\mathit{in}(R) \cap \mathit{out}(S)) + & \{\!\![\text{ distributivity }]\!\!\} \\ & (\mathit{in}(R) \cap \mathit{hid}(S)) + (\mathit{out}(R) \cap \mathit{in}(S)) + & \\ & (\mathit{out}(R) \cap \mathit{out}(S)) + (\mathit{out}(R) \cap \mathit{hid}(S)) + & \\ & (\mathit{hid}(R) \cap \mathit{in}(S)) + (\mathit{hid}(R) \cap \mathit{out}(S)) + & \\ & (\mathit{hid}(R) \cap \mathit{hid}(S)) & \\ = & \{\} + (\mathit{in}(R) \cap \mathit{out}(S)) + & \{\!\![\text{ Parts 1 and 2 }]\!\!\} \\ & \{\} + (\mathit{out}(R) \cap \mathit{in}(S)) + & \\ & \{\} + \{\} + \{\} + \{\} + \{\} & \\ = & (\mathit{in}(R) \cap \mathit{out}(S)) \cup (\mathit{in}(S) \cap \mathit{out}(R)) \ . & \{\!\![\text{ neutrality of } \{\}\ ]\!\!\} \end{array}$$

4. By the definitions, (7) and point-disjointness of R, S with Boolean algebra:

$$\mathit{in}(R) \cap \mathit{out}(S) = \neg PRP \cap QS\neg Q = \neg PQ(R \cap S)P\neg Q = Q(R \cap S)P \ .$$

   The second equation is proved symmetrically.

5. By the definitions and distributivity:

$$\begin{aligned} \mathit{hid}(R + S) &= (P + Q)(R + S)(P + Q) \\ &= (P + Q)R(P + Q)(P + Q)S(P + Q) \ . \end{aligned}$$

   We only continue with the first summand, the second being symmetric.

$$\begin{array}{lll} & (P + Q)R(P + Q) & \\ = & PRP + PRQ + QRP + QRQ & \{\!\![\text{ distributivity }]\!\!\} \\ = & \mathit{hid}(R) + PRQ + QRP + \{\} & \{\!\![\text{ the definitions with Part 2 }]\!\!\} \\ \supseteq & \mathit{hid}(R) + P(R \cap S)Q + Q(R \cap S)P & \{\!\![\text{ neutrality of } \{\} \text{ with} \\ & & \quad \text{Boolean algebra }]\!\!\} \\ = & \mathit{hid}(R) + (\mathit{in}(R) \cap \mathit{out}(S)) + & \{\!\![\text{ Part 2 }]\!\!\} \\ & (\mathit{out}(R) \cap \mathit{in}(S)) & \\ = & \mathit{hid}(R) + R \cap S \ . & \{\!\![\text{ Part 4 }]\!\!\} \end{array}$$

   The analogous calculation for the second summand yields $\mathit{hid}(R + S) \supseteq \mathit{hid}(S)$, which establishes the claim.

□

Proof of Lm. 6.3.

1. Let $P =_{df} \mathit{points}(R)$, $Q =_{df} \mathit{points}(S)$, $A =_{df} \mathit{arrows}(R)$ and $A =_{df} \mathit{arrows}(R)$. We only show the first claim, the second being symmetric.

$$
\begin{array}{ll}
 & \mathit{in}(R+S) \\
= & \quad \{\!\![\text{ definition of } \mathit{in} \ ]\!\!\} \\
 & \neg(P+Q)(A \cup B)(P+Q) \\
= & \quad \{\!\![\text{ relation algebra }]\!\!\} \\
 & \neg P \neg Q(AP \cup AQ \cup BP \cup BQ) \\
= & \quad \{\!\![\text{ distributivity and commutativity of point sets }]\!\!\} \\
 & \neg Q \neg PAP \cup \neg Q \neg PAQ \cup \neg P \neg QBP \cup \neg P \neg QBQ \\
= & \quad \{\!\![\text{ definition of } \mathit{in} \text{ with } Q \subseteq \neg P,\ P \subseteq \neg Q \text{ and loose-freeness of } R,S \ ]\!\!\} \\
 & \neg Q\mathit{in}(R) \cup \{\} \cup \{\} \cup \neg P\mathit{in}(S)\ .
\end{array}
$$

To determine the first summand we calculate

$$
\begin{array}{ll}
 & \mathit{in}(R) \\
= & \quad \{\!\![\text{ neutrality of } I \ ]\!\!\} \\
 & I\,\mathit{in}(R) \\
= & \quad \{\!\![\text{ relative complements }]\!\!\} \\
 & (Q + \neg Q)\mathit{in}(R) \\
= & \quad \{\!\![\text{ distributivity and definition of } \mathit{in} \ ]\!\!\} \\
 & Q\neg PAP + \neg Q\mathit{in}(R) \\
= & \quad \{\!\![\ Q \subseteq \neg P \ ]\!\!\} \\
 & QAP + \neg Q\mathit{in}(R) \\
= & \quad \{\!\![\ R,S \text{ arrow-closed and Lm. 4.5.2 }]\!\!\} \\
 & QBP + \neg Q\mathit{in}(R)\ .
\end{array}
$$

By Boolean algebra this is equivalent to $\neg Q\mathit{in}(R) = \mathit{in}(R) - QBP$. For this we calculate further:

$$
\begin{array}{ll}
 & \mathit{in}(R) - QBP \\
= & \quad \{\!\![\ P \subseteq \neg Q \text{ and lattice algebra }]\!\!\} \\
 & \mathit{in}(R) - QB\neg QP \\
= & \quad \{\!\![\text{ definition of } \mathit{out} \ ]\!\!\} \\
 & \mathit{in}(R) - \mathit{out}(S)P \\
= & \quad \{\!\![\text{ definition of difference }]\!\!\} \\
 & \mathit{in}(R) \cap \overline{\mathit{out}(S)P} \\
= & \quad \{\!\![\text{ complement of restriction }]\!\!\} \\
 & \mathit{in}(R) \cap (\overline{\mathit{out}(S)} \cup \mathit{out}(S)\neg P) \\
= & \quad \{\!\![\text{ distributivity }]\!\!\} \\
 & (\mathit{in}(R) \cap \overline{\mathit{out}(S)}) \cup (\mathit{in}(R) \cap \mathit{out}(S)\neg P) \\
= & \quad \{\!\![\text{ definition of difference and } \mathit{in} \ ]\!\!\}
\end{array}
$$

$$\begin{array}{ll} & (in(R) - out(S)) \cup (\neg PAP \cap out(S)\neg P) \\ = & \quad \{\!\![\text{ restriction property (7) }]\!\!\} \\ & (in(R) - out(S)) \cup \{\} \ . \end{array}$$

Altogether we have proved $\neg Q\,in(R) = in(R) - out(S)$. Symmetrically one shows $\neg P\,in(S) = in(S) - out(R)$, which establishes the claim.

2. Again we only show the first claim, the second being symmetric.

$$\begin{array}{ll} & (in(R) + in(S)) - (out(R) + out(S)) \\ = & \quad \{\!\![\text{ definition of difference }]\!\!\} \\ & (in(R) + in(S)) \cap \overline{out(R) + out(S)} \\ = & \quad \{\!\![\text{ Boolean algebra }]\!\!\} \\ & (in(R) + in(S)) \cap \overline{out(R)} \cap \overline{out(S)} \\ = & \quad \{\!\![\text{ distributivity }]\!\!\} \\ & (in(R) \cap \overline{out(R)} \cap \overline{out(S)}) + (in(S) \cap \overline{out(R)} \cap \overline{out(S)}) \\ = & \quad \{\!\![\text{ Boolean algebra }]\!\!\} \\ & (in(R) - out(R) - out(S)) + (in(S) - out(S) - out(R)) \\ = & \quad \{\!\![\text{ disjointness of } in \text{ and } out \text{ of a graphlet (Def. 5.2) }]\!\!\} \\ & (in(R) - out(S)) + (in(S) - out(R)) \ . \end{array}$$

Now the claim follows from Part 1.

3.
$$\begin{array}{ll} & perimeter(R)\,\Delta\,perimeter(S) \\ = & \quad \{\!\![\text{ definition of symmetric difference }]\!\!\} \\ & (perimeter(R) - perimeter(S)) \cup (perimeter(S) - perimeter(R)) \\ = & \quad \{\!\![\text{ definition of perimeter }]\!\!\} \\ & ((in(R) + out(R)) - (in(S) + out(S)))+ \\ & ((in(S) + out(S)) - (in(R) + out(R))) \\ = & \quad \{\!\![\text{ Boolean algebra }]\!\!\} \\ & (in(R) - in(S) - out(S)) + (out(R) - in(S) - out(S))+ \\ & (in(S) - in(R) - out(R)) + (out(S) - in(R) - out(R)) \\ = & \quad \{\!\![\text{ by Lm. 5.4.1 and Boolean algebra }]\!\!\} \\ & (in(R) - out(S)) + (out(R) - in(S))+ \\ & (in(S) - out(R)) + (out(S) - in(R)) \\ = & \quad \{\!\![\text{ by Part 1 and Boolean algebra }]\!\!\} \\ & in(R + S) + out(S + R) \\ = & \quad \{\!\![\text{ definition of perimeter }]\!\!\} \\ & perimeter(R + S) \ . \end{array}$$

□

Proof of Lm. 7.1. Let S be acyclic and fork-free. By properties of * and $^+$, the first Dedekind rule, fork-freeness and isotony, properties of * and $^+$ and acyclicity of S with strictness:

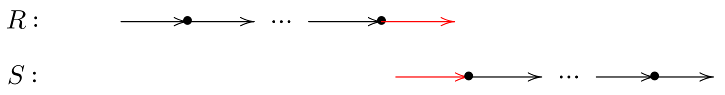
$$\begin{aligned} S^+S^+ \cap S \ &= \ SS^*S^+ \cap S \ \subseteq \ S(S^*S^+ \cap S^{\smile}S) \\ &\subseteq \ S(S^*S^+ \cap I) \ = \ S(S^+ \cap I) \ = \ \{\} \ . \end{aligned}$$

The proof for join-freeness is dual. □

Proof of Th. 7.3.
Let $points(R) = [u,v]$ and $points(S) = [x,y]$ with $[u,v] \cap [x,y] = \{\}$ and assume the definedness conditions for $R \gg S$. The property $points(R \gg S) = points(R) \cup points(S)$ is immediate from $points(R \gg S) = (R \cup S) \cap I$ and distributivity; it also entails $in(R \gg S) = in(R)$ and $out(R \gg S) = out(S)$.

To show that $R \gg S$ is a line we first derive a detailed representation of $(R \cup S)^+$. For abbreviation we set $iR =_{df} in(R)$ and $oR =_{df} out(R)$, and similarly for S, T.

By $v = tail(oR)$ and $x = head(iS)$ we infer from $oR = iS$ and $oS \cap iR = \{\}$, together with fork-freeness of R and join-freeness of S, that $oR = iS = \{(v,x)\} = T$, where $T =_{df} R \cap S$. Moreover, R and S satisfy $SR = \{\}$ and $RS = RT \cup TS$. In the following diagram T is drawn in red:



From Lm. 16.4.2 we infer $(R \cup S)^+ = R^+ \cup R^+S^+ \cup S^+$. We continue with the middle summand. By definition of plus, above property $RS = RT \cup TS$ and distibutivity, definition of plus, splitting R^* and S^* with distributivity and neutrality, idempotence of union

$$\begin{aligned} R^+S^+ &= R^*RSS^* = R^*RTS^* \cup R^*TSS^* = R^+TS^* \cup R^*TS^+ \\ &= R^+TS^+ \cup R^+T \cup R^+TS^+ \cup TS^+ = R^+T \cup R^+TS^+ \cup TS^+ \ . \end{aligned}$$

Thus,

$$(R \cup S)^+ \ = \ R^+ \cup R^+S^+ \cup S^+ \ = \ R^+ \cup R^+T \cup R^+TS^+ \cup TS^+ \cup S^+ \ ,$$

which simplifies to $R^+ \cup R^+TS^+ \cup S^+$, since by definition of T we have $T \subseteq R$ and $T \subseteq S$ and therefore $R^+T \subseteq R^+$ and $TS^+ \subseteq S^+$. Altogether,

$$(R \cup S)^+ = R^+ \cup R^+TS^+ \cup S^+ \ . \tag{9}$$

Now we can prove acyclicity of $R \cup S$ by showing, according to Lm. 16.2, asymmetry of $(R \cup S)^+$. By (9) and distributivity,

$$\begin{aligned} &(R \cup S)^+ \cap ((R \cup S)^+)^{\smile} \\ &\quad = (R^+ \cup R^+TS^+ \cup S^+) \cap (R^+ \cup R^+TS^+ \cup S^+)^{\smile} \\ &\quad = (R^+ \cap (R^+)^{\smile}) \cup (R^+ \cap (S^+)^{\smile}T^{\smile}(R^+)^{\smile}) \\ &\qquad \cup (R^+ \cap (S^+)^{\smile}) \cup (R^+TS^+ \cap (R^+)^{\smile}) \\ &\qquad \cup (R^+TS^+ \cap (S^+)^{\smile}T^{\smile}(R^+)^{\smile}) \\ &\qquad \cup (R^+TS^+ \cap (S^+)^{\smile}) \cup (S^+ \cap (R^+)^{\smile}) \\ &\qquad \cup (S^+ \cap (S^+)^{\smile}T^{\smile}(R^+)^{\smile}) \cup (S^+ \cap (S^+)^{\smile}) \ . \end{aligned}$$

The first and last summands are $\{\}$ by acyclicity and hence asymmetry of R and S. For the remaining ones we use that the intersection of two graphlets is empty if the intersection of their *heads* or of their *tails* is empty. For instance, using the laws (8),

$$tails(R^+ T S^+) \cap tails((S^+)^{\smile} T^{\smile} (R^+)^{\smile}) \subseteq tails(R^+) \cap tails((S^+)^{\smile}) = tails(R) \cap heads(S) = \{\} \ .$$

For fork-freeness (join-freeness is symmetric) we use

$$(R \cup S)^{\smile} (R \cup S)^{\smile} = R^{\smile} R \cup R^{\smile} S \cup S^{\smile} R \cup S^{\smile} S \ .$$

The first and last summands are $\subseteq I$, since R, S are lines. For the second summand we have $x\,(R^{\smile} S)\,y$ iff there is a z with $z\,R\,x$ and $z\,S\,y$. This implies $z \in tails(R) \cap tails(S)$. By the definitions, for open lines this intersection is $\{greatest(R)\}$, and the only arrow starting in that point is the unique one in $oR \cap iS$; hence $x = y$. Hence $R^{\smile} S \subseteq I$, as it should be. The third summand is the converse of the second one and therefore also $\subseteq I$.

Connectedness holds, since by (9) it is immediate that $points(R \gg S) = [u, y]$, where the interval is taken w.r.t. $R \cup S$.

Finally we show associativity. We only need to check that $R \gg (S \gg T)$ and $(R \gg S) \gg T$ have the same assumptions. Assume that R, S, T are arbitrary pairwise point-disjoint graphlets and $R \gg (S \gg T)$ and $(R \gg S) \gg T$ are defined. We calculate as follows.

$$\begin{array}{ll}
 & assu(R \gg (S \gg T)) \\
\Leftrightarrow & \{\!\![\text{ definitions }]\!\!\} \\
 & oS = iT \ \wedge \ oT \cap iS = \{\} \wedge \\
 & oR = in(S \gg T) \ \wedge \ out(S \gg T) \cap iR = \{\} \\
\Leftrightarrow & \{\!\![\text{ above formulas for } in \text{ and } out \ ]\!\!\} \\
 & oS = iT \ \wedge \ oT \cap iS = \{\} \wedge \\
 & oR = iS \ \wedge \ oT \cap iR = \{\} \\
\Leftrightarrow & \{\!\![\text{ logic }]\!\!\} \\
 & oR = iS \ \wedge \ oS = iT \ \wedge \ oT \cap iR = \{\} \ \wedge \ oT \cap iS = \{\} \\
\Leftrightarrow & \{\!\![\text{ inputs and outputs of open lines are singletons }]\!\!\} \\
 & oR = iS \ \wedge \ oS = iT \ \wedge \ (oS \cap iR = \{\} \ \vee \ oS = iR) \wedge \\
 & oT \cap iR = \{\} \ \wedge \ oT \cap iS = \{\} \\
\Leftrightarrow & \{\!\![\text{ distributivity }]\!\!\} \\
 & ((oR = iS \ \wedge \ oS = iT \ \wedge \ oS \cap iR = \{\}) \vee \\
 & (oR = iS \ \wedge \ oS = iT \ \wedge \ oS = iR)) \wedge \\
 & oT \cap iR = \{\} \ \wedge \ oT \cap iS = \{\} \\
\Leftrightarrow & \{\!\![\text{ logic }]\!\!\} \\
 & ((oR = iS \ \wedge \ oS = iT \ \wedge \ oS \cap iR = \{\}) \vee \\
 & (oR = iS \ \wedge \ oS = iT \ \wedge \ oS = iR \ \wedge \ iR = iT)) \wedge \\
 & oT \cap (iR \cup iS) = \{\} \\
\Leftrightarrow & \{\!\![\text{ second disjunct false by Lm. 5.4.6 }]\!\!\} \\
 & oR = iS \ \wedge \ oS = iT \ \wedge \ oS \cap iR = \{\} \ \wedge \ oT \cap iR = \{\} \ \wedge \ oT \cap iS = \{\} \ .
\end{array}$$

A symmetric derivation hows that also $assu((R \gg S) \gg T)$ is equivalent to that formula, and we are done. □

Proof of Th. 11.2.

1. For reflexivity we reason as follows. Assume $assu(p) \wedge claim(p)$. Then reflexivity of $\sqsubseteq$ establishes $p \leq p$.
   For transitivity we suppose $p \leq q$ and $q \leq r$, and assume $assu(p)$.
   - From $p \leq q$ we infer $assu(q)$.
   - From $q \leq r$ we infer $assu(q)$.
   - Now we assume $claim(r)$.
   - From $q \leq r$ we infer $claim(q) \wedge q \sqsubseteq r$.
   - From $p \leq q$ we infer $claim(r) \wedge p \sqsubseteq q$.
   - By transitivity of $\sqsubseteq$ we infer $p \sqsubseteq r$.
2. By definition of $\sim$, $p \leq q$ holds trivially if $\neg\, assu(p)$. The second claim holds for arbitrary lax orders and their least elements.
3. By Part 2 it suffices to show $p \leq q$ for every non-erroneous p and faulty q. However, this is immediate from the definition of $\leq$. The second claim holds for arbitrary preorders and their least elements.
4. From Part 2 we know $\bot \leq \bot \circ p$. Moreover, the definition of *assu* tells us that $assu(\bot \circ p)$ is false, since $assu(\bot)$ is false, and again by Part 2 we obtain also $\bot \circ p \leq p$.
   The second equivalence is shown analogously.
5. From Part 3 we know $\top \circ p \leq \top$. Assume now that $\top \circ p$ is non-erroneous, i.e., $assu(\top)$. From the definition of *claim* we infer
   $$claim(\top \circ p) \Leftrightarrow claim(\top) \wedge claim(p) \wedge \circ\text{-}claim(\top \circ p) \Leftrightarrow \mathsf{false}\ ,$$
   since $claim(\top)$ is false by definition. Therefore $\top \leq \top \circ p$ is immediate from the definition of $\leq$.
   The second equivalence is shown analogously.
6. Immediate from (5).

□

These proofs are also readily found automatically by Prover9 [19].

Proof of Th. 12.4. We only show the first claim; the second one is symmetric.

Assume $assu(\text{lhs})$, otherwise the refinement holds trivially. Then also $assu(p)$, $assu(q)$ and $assu(r)$. Then

$$\begin{array}{ll}
& \mathsf{TRUE} \\
\Leftrightarrow & \{\!\![\ \text{definitions of } \circ_1 \text{ and } \circ_2\ ]\!\!\} \\
& \circ_2\text{-}assu(p,q) \wedge \circ_1\text{-}assu(p+q,R) \\
\Leftrightarrow & \{\!\![\ \text{modularity of } \circ_1\text{-}assu\ ]\!\!\} \\
& \circ_2\text{-}assu(p,q) \wedge \circ_1\text{-}assu(p,r) \wedge \circ_1\text{-}assu(q,r) \\
\Rightarrow & \{\!\![\ \text{assumption } \circ_1\text{-}assu \Rightarrow \circ_2\text{-}assu\ ]\!\!\} \\
& \circ_2\text{-}assu(p,q) \wedge \circ_2\text{-}assu(p,r) \wedge \circ_1\text{-}assu(q,r) \\
\Leftrightarrow & \{\!\![\ \text{modularity of } \circ_2\text{-}assu\ ]\!\!\} \\
& \circ_2\text{-}assu(p,q+r) \wedge \circ_1\text{-}assu(q,r)\ .
\end{array}$$

From this we infer $assu(\text{rhs})$. Now assume $claim(\text{rhs})$, otherwise the refinement holds trivially. By the definitions and associativity of $+$ we have

$$val(\text{lhs}) = (val(p) + val(q)) + val(r) = val(p) + (val(q) + val(r) = val(\text{rhs})$$

and hence also $claim(\text{lhs}) \Leftrightarrow claim(\text{rhs})$.

Moreover,

$$\begin{array}{ll}
 & com(\text{lhs}) \\
= & \quad \{\!\![\text{ definitions }]\!\!\} \\
 & com(p) \cup com(q) \cup com(r) \cup \circ_2\text{-}com(p,q) \cup \circ_1\text{-}com(p+q,r) \\
= & \quad \{\!\![\text{ modularity of } \circ_1\text{-}com \ ]\!\!\} \\
 & com(p) \cup com(q) \cup com(r) \cup \circ_2\text{-}com(p,q) \cup \circ_1\text{-}com(p,r) \cup \circ_1\text{-}com(q,r) \\
\subseteq & \quad \{\!\![\text{ assumption } \circ_1\text{-}com(p,r) \subseteq \circ_2\text{-}com(p,r) \ ]\!\!\} \\
 & com(p) \cup com(q) \cup com(r) \cup \circ_2\text{-}com(p,q) \cup \circ_2\text{-}com(p,r) \cup \circ_1\text{-}com(q,r) \\
= & \quad \{\!\![\text{ modularity of } \circ_2\text{-}com \ ]\!\!\} \\
 & com(p) \cup com(q) \cup com(r) \cup \circ_2\text{-}com(p,q+r) \cup \circ_1\text{-}com(q,r) \\
= & \quad \{\!\![\text{ associativity and commutativity of } \cup \text{ and definitions }]\!\!\} \\
 & com(\text{rhs}) \ .
\end{array}$$

This establishes the claim. □

Proof of Cor. 12.5. Immediate from Th. 12.4 setting $\circ_2 =_{df} \circ_1$. □

Proof of Th. 12.6. Assume $assu(\text{lhs})$, otherwise the refinement holds trivially. This implies $assu(p)$, $assu(q)$, $assu(r)$ and $assu(s)$. Then

$$\begin{array}{ll}
 & \mathsf{TRUE} \\
\Leftrightarrow & \quad \{\!\![\text{ definitions of } \circ_1 \text{ and } \circ_2 \ ]\!\!\} \\
 & \circ_2\text{-}assu(p,q) \wedge \circ_2\text{-}assu(r,s) \wedge \circ_1\text{-}assu(p \oplus q, r \oplus s) \\
\Leftrightarrow & \quad \{\!\![\text{ modularity of } \circ_1\text{-}assu \ ]\!\!\} \\
 & \circ_2\text{-}assu(p,q) \wedge \circ_2\text{-}assu(r,s) \wedge \circ_1\text{-}assu(p,r) \wedge \circ_1\text{-}assu(p,s) \wedge \\
 & \circ_1\text{-}assu(q,r) \wedge \circ_1\text{-}assu(q,s) \\
\Leftrightarrow & \quad \{\!\![\text{ associativity and commutativity of } \wedge \ ]\!\!\} \\
 & \circ_1\text{-}assu(p,r) \wedge \circ_1\text{-}assu(q,s) \wedge \circ_2\text{-}assu(p,q) \wedge \circ_1\text{-}assu(p,s) \wedge \\
 & \circ_1\text{-}assu(q,r) \wedge \circ_2\text{-}assu(r,s) \\
\Rightarrow & \quad \{\!\![\text{ assumption } \circ_1\text{-}assu \Rightarrow \circ_2\text{-}assu \ ]\!\!\} \\
 & \circ_1\text{-}assu(p,r) \wedge \circ_1\text{-}assu(q,s) \wedge \circ_2\text{-}assu(p,q) \wedge \circ_2\text{-}assu(p,s) \wedge \\
 & \circ_2\text{-}assu(q,r) \wedge \circ_2\text{-}assu(r,s) \\
\Rightarrow & \quad \{\!\![\text{ assumed symmetry of } \circ_2\text{-}assu \ ]\!\!\} \\
 & \circ_1\text{-}assu(p,r) \wedge \circ_1\text{-}assu(q,s) \wedge \circ_2\text{-}assu(p,q) \wedge \circ_2\text{-}assu(p,s) \wedge \\
 & \circ_2\text{-}assu(R,Q) \wedge \circ_2\text{-}assu(r,s) \\
\Leftrightarrow & \quad \{\!\![\text{ modularity of } \circ_2\text{-}assu \ ]\!\!\} \\
 & \circ_1\text{-}assu(p,r) \wedge \circ_1\text{-}assu(q,s) \wedge \circ_2\text{-}assu(p \oplus r, q \oplus s) \ .
\end{array}$$

From this we infer $assu(rhs)$. Now assume $claim(\text{rhs})$, otherwise the refinement holds trivially. By the definitions and associativity and commutativity of $+$,

$$\begin{aligned} val(\text{lhs}) &= (val(p) + val(q)) + (val(r) + val(s)) \\ &= (val(p) + val(r)) + (val(q) + val(s)) = val(\text{rhs}) \end{aligned}$$

and hence also $claim(\text{lhs}) \Leftrightarrow claim(\text{rhs})$. Moreover,

$com(\text{lhs})$
$=$ $\{\!\![$ definitions and associativity and commutativity of $+$ $]\!\!\}$
$com(p) \cup com(q) \cup com(r) \cup com(s) \cup$
$\circ_2\text{-}com(p,q) \cup \circ_2\text{-}com(r,s) \cup \circ_1\text{-}com(p \oplus q, r \oplus s)$
$\subseteq$ $\{\!\![$ setting $t =_{df} com(p) \cup com(q) \cup com(r) \cup com(s)$ and submodularity of $\circ_1\text{-}com$ $]\!\!\}$
$t \cup \circ_2\text{-}com(p,q) \cup \circ_2\text{-}com(r,s) \cup \circ_1\text{-}com(p,r) \cup \circ_1\text{-}com(p,s) \cup$
$\circ_1\text{-}com(Q,R) \cup \circ_1\text{-}com(q,s)$
$=$ $\{\!\![$ associativity and commutativity of $\cup$ $]\!\!\}$
$t \cup \circ_1\text{-}com(p,r) \cup \circ_1\text{-}com(q,s) \cup \circ_2\text{-}com(p,q) \cup \circ_1\text{-}com(p,s) \cup$
$\circ_1\text{-}com(Q,R) \cup \circ_2\text{-}com(r,s)$
$\subseteq$ $\{\!\![$ assumption $\circ_1\text{-}com(p,s) \subseteq \circ_2\text{-}com(p,s)$ $]\!\!\}$
$t \cup \circ_1\text{-}com(p,r) \cup \circ_1\text{-}com(q,s) \cup \circ_2\text{-}com(p,q) \cup \circ_2\text{-}com(p,s) \cup$
$\circ_2\text{-}com(Q,R) \cup \circ_2\text{-}com(r,s)$
$=$ $\{\!\![$ assumed commutativity of $\circ_2\text{-}com$ $]\!\!\}$
$t \cup \circ_1\text{-}com(p,r) \cup \circ_1\text{-}com(q,s) \cup \circ_2\text{-}com(p,q) \cup \circ_2\text{-}com(p,s) \cup$
$\circ_2\text{-}com(R,Q) \cup \circ_2\text{-}com(r,s)$
$=$ $\{\!\![$ modularity of $\circ_2\text{-}com$ $]\!\!\}$
$t \cup \circ_1\text{-}com(p,r) \cup \circ_1\text{-}com(q,s) \cup \circ_2\text{-}com(p \oplus r, q \oplus s)$.
$=$ $\{\!\![$ associativity and commutativity of $\cup$ and definitions $]\!\!\}$
$com(\text{rhs})$.

This establishes the claim. $\square$

Proof of Th. 13.2. We show modularity of ;-$assu$ in its left argument; the case of the right argument is symmetric.

For the proof it is convenient to transform ;-$assu$. We use the shorthands $P =_{df} points(R), Q =_{df} points(S)$ and $A =_{df} arrows(R), A =_{df} arrows(R)$.

$in(R) \cap out(S)$
$=$ $\{\!\![$ definitions $]\!\!\}$
$\neg PAP \cap QB\neg Q$
$=$ $\{\!\![$ restriction (7) and commutativity of point sets $]\!\!\}$
$\neg PQAP \cap QB\neg QP$
$\Leftrightarrow$ $\{\!\![$ point-disjointness of R and S and Boolean algebra $]\!\!\}$
$QAP \cap QBP$
$=$ $\{\!\![$ restriction (7) $]\!\!\}$

$Q(A \cap B)P$.

Hence ;-$assu(R,S) \Leftrightarrow Q(A \cap B)P \cap \mathsf{Dep} = \{\}$. Now we can adapt the proof of Theorem 3.6 in [20] for our purposes, setting $V =_{df} points(T)$ and $C =_{df} arrows(T)$:

;-$assu(R+S,T)$
$\Leftrightarrow$ {[definitions]}
$V((A \cup B) \cap C)(P+Q) \cap \mathsf{Dep} = \{\}$
$\Leftrightarrow$ {[distributivity, set algebra]}
$V(A \cap C)P \cap \mathsf{Dep} = \{\} \wedge V(A \cap C)Q \cap \mathsf{Dep} = \{\} \wedge$
$V(B \cap C)P \cap \mathsf{Dep} = \{\} \wedge V(B \cap C)Q \cap \mathsf{Dep} = \{\}$
$\Leftrightarrow$ {[by the definitions, and since $Q \cap P = P \cap V = Q \cap V = \{\}$, hence $V, Q \subseteq \neg P$ and $V, P \subseteq \neg Q$, therefore, by the assumption of no loose arrows, $VAQ = VBP = \{\}$]}
;-$assu(R,T) \wedge \mathsf{TRUE} \wedge \mathsf{TRUE} \wedge$;-$assu(S,T)$. □

Proof of Lm. 13.5.

1. $in(R_1 \mid\mid\mid R_2)$
   $=$ {[definition of $\mid\mid\mid$]}
   $in(R_1 + R_2)$
   $=$ {[Lm. 6.3.1]}
   $(in(R_1) - out(R_2)) + (in(R_2) - out(R_1))$
   $=$ {[+-$assu(R_1, R_2)$ with Boolean algebra]}
   $in(R_1) + in(R_2)$.
2. Analogous to Part 1.
3. $hid(R_1 \mid\mid\mid R_2)$
   $=$ {[decomposition property (4) and Boolean algebra]}
   $arrows(R_1 \mid\mid\mid R_2) - in(R_1 \mid\mid\mid R_2) - out(R_1 \mid\mid\mid R_2)$
   $=$ {[Parts 1 and 2]}
   $(arrows(R_1) \cup arrows(R_2)) - (in(R_1) + in(R_2)) - (out(R_1) + out(R_2))$
   $=$ {[Boolean algebra]}
   $(arrows(R_1) - in(R_1) - in(R_2) - out(R_1) - out(R_2)) \cup$
   $(arrows(R_2) - in(R_1) - in(R_2) - out(R_1) - out(R_2))$
   $=$ {[Boolean algebra]}
   $(arrows(R_1) - in(R_1) - out(R_1) - in(R_2) - out(R_2)) \cup$
   $(arrows(R_2) - in(R_2) - out(R_2) - in(R_1) - out(R_1))$
   $=$ {[decomposition property (4) with Boolean algebra]}
   $(hid(R_1) - in(R_2) - out(R_2)) \cup (hid(R_2) - in(R_1) - out(R_1))$
   $=$ {[Lm. 5.4.1 and Lm. 5.4.2 with Boolean algebra]}
   $hid(R_1) + hid(R_2)$.

□