Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 109 (2004) 3–15
www.elsevier.com/locate/entcs

On the Visual Representation of Configuration in Reconfigurable Computing

Phan C. Vinh [1] and Jonathan P. Bowen [2]

Research Institute for Computing
London South Bank University
London, UK

Abstract

In this paper we aim to show formally a visual representation of a regular array structure-based logical configuration in reconfigurable computing by using the clear syntax and formal semantics. In other words, some particular types of objects satisfying certain conditions will define a logical configuration; this is the syntax of our representation. We also consider which arrangements of objects in a given logical configuration will be formally defined; this is the semantics. The rules for reasoning about changing a logical configuration are formulated. Subsequently, their soundness is proven. A logical configuration is provable from another one by applying these rules.

Keywords: Visual representation, Graph-based approach, Reconfigurable computing, Regular array structure.

1 Introduction

One of the topics arising from the study of logics for design automation is the visual representation of configuration by mathematical concepts in reconfigurable computing. A natural question, then, is whether or not visual reasoning concepts can be formalized in a way that preserves their inherently visual nature. The answer to this question is that they can, as will be demonstrated in this paper. This reasoning is based a precisely defined syntax and semantics of visual configuration. We will define a configuration, which is composed of

[1] Email: phanvc@lsbu.ac.uk
[2] Email: [illegible]@lsbu.ac.uk

1571-0661
doi:10.1016/j.entcs.2004.02.052

particular types of objects satisfying certain conditions as the syntax of our representation. We will also give a formal definition of which arrangements of objects in a given configuration as the semantics. Subsequently, we will give precise rules for manipulating the configurations. In order to work with our configurations, we choose the regular array structure as a particular one to construct the configuration. We will have to decide which of their features are meaningful, and which are not. A crucial idea will be that all of the meaningful information given by a configuration is contained in its topology, i.e. the general arrangement of its objects. Another way of saying this is that if one configuration can be transformed into another by transformation rules, then the two configurations are essentially the same. This is typical of formally logical reasoning in general.

2 The Regular Array Structure Model

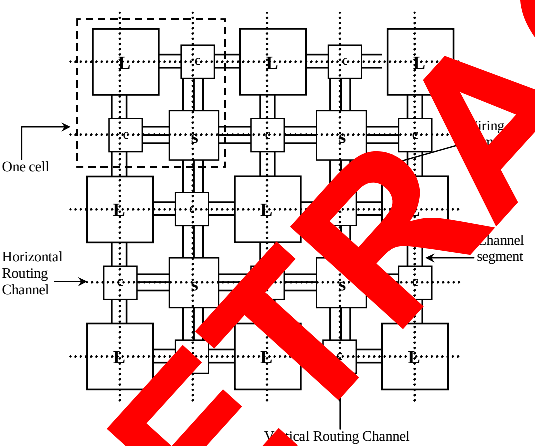


Fig. 1. A regular array structure model

For representing the configuration, an appropriate model must be defined. In reconfigurable computing area the representation and reasoning are usually carried out on a regular array structure, therefore the chosen model for abstracting consists of a two-dimensional array of logic cells interconnected by vertical and horizontal routing channels [3] as shown in Figure 1. The model comprises three major parts: the Logic blocks (L), Connection blocks (C), and Switch blocks (S). The logic blocks house the combinational and sequential logic that form the functionality of an operation. The C blocks are rectangular switch boxes with connection points on all four sides, and are used to connect the logic block pins to the routing channels, via programmable switches. The S blocks are also rectangular switch boxes. They are used to connect wiring segments in one channel segment to those in another. The two-dimensional grid that is overlaid on the regular array structure is used in this paper as a means of describing the connections to be routed.

3 Related Work

Graphs are usually associated with intuitions and illustrations, not with rigorous proofs. Visual representations are allowed in the context of discovery, not in the context of justification, in which empirical justifications have used for graphs instead of analytical justification. Thus, there are several mistakes related to the use of graphs, one of them being the reliance on graphs to guide the logic in the construction of proofs instead of the axioms [5]. This implies that the use of pictures is a flaw in a formal system. R. Bardohl et al. [2], Isabel Luengo [1] and Miller [4] have shown that the problem is not with graphs, but with having bad semantics and syntax. They have also determined that a graph-based reasoning system can be built for graphs with formal semantics, syntax and rules of inference, and that it is a sound system, meaning that no fallacies can be derived from it. The fallacies in graphs arise from the fact that the accidental features of the graph are taken to represent features. This is why a system with clear syntax and semantics will help make fallacies impossible.

4 Motivation

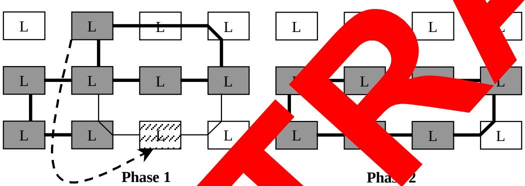


Fig. 2. Two-phase dynamic relocation procedure of Logic blocks

In regular array structure, any online management strategy implies a dynamic relocation mechanism of the available Logic resources (L), whereby the system tries to avoid a lack of contiguous free L resources from preventing the configuration of new functions(provided that the total number of L resources available is sufficient). If a new function cannot be allocated immediately due to lack of contiguous free L resources, a suitable rearrangement of a subset of the functions currently running can solve the problem. Any reconfiguration action must therefore ensure that the links from the original L are not broken before being totally re-established from its replica; otherwise its operation will be disturbed or even halted. The possible solution is to divide the relocation procedure in two phases, as illustrated in Figure 2. In the first phase, the configuration of the L is copied into the new location and the links of both Ls are placed in parallel. In the second phase, when the links of the L replica are already perfectly stable, the original L and its links are freed from the configuration of circuit. We manipulate the rules, as the deductive principles, to arrange formally the logical configuration representing an application on the running regular array structure.

5 Syntax

5.1 Objects

There are two different classes of configurational objects: primitive and derived. The primitive objects are not defined. The derived objects are defined in terms of the primitive objects.

5.1.1 Primitive Objects

(*i*) **Frame:** A frame is a regular array structure with dashed edges bounded inside four edges (East, West, South, North)of its border; see Figure 3.

(*ii*) **Cell:** A cell is a small square cell, represented by □, we will use A, B, C, with superscripts and subscripts as variables over cell.

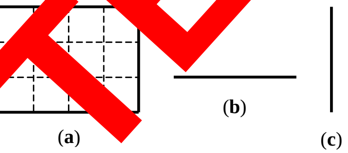


Fig. 3. (**a**)-A frame, (**b**)-A row, (**c**)-A column

(*iii*) **Row:** A row is a horizontal straight edge; see Figure 3.

(*iv*) **Column A:** column is a vertical straight edge; see Figure 3.

We will use l, m, n, with superscripts and subscripts as variables over row (column) by indicating explicitly 'row' ('column')

5.1.2 Relations

(*i*) *In* $\subseteq$ objects × frame: A configurational object is *in* a frame iff none of parts of the object extends outside the frame.

(*ii*) *On* $\subseteq$ cell × row (column): A cell is *on* a row (column) iff they intersect. We will also say that a row (column) l *goes through* a cell A if A is *on* l.

5.1.3 Derived Objects

(*i*) **Section and semi-section:** A section consists of two distinct cells on a row (column) l and the part of l that lies between them regardless if any other cells between them. The section defined by cells A and B is called $[A,B]$ or $[B,A]$. See Figure 4.

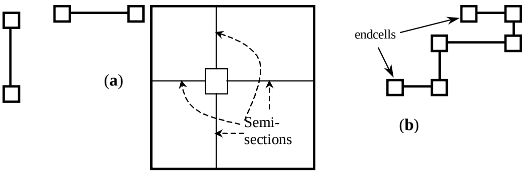


Fig. 4. (**a**)-Sections and semi-sections, (**b**)-A route

A semi-section of a row (column) l is the part of l lying between a cell on row (column) and border of frame regardless if any other cells between them. The semi-section defined by cell A and west side of border is called $[A,\text{West})$ or $(\text{West},A]$. See Figure 4.
(ii) **Route:** A route is a sequence of connected sections. See Figure 4.

5.1.4 Relations

(i)*Intersects* $\subseteq$ cell $\times$ route: A cell *intersects* a route iff it is one of two distinct cells determining a section of route. Note that each cell on section of route intersects route once or twice, and any of them, which intersects route once is said *endcell* of route. Given any two endcells, there exists *at least* a route intersects them and due to finiteness of the objects in configuration, there also exists at least a route only including *at most* two sections (called **mroute** for short). The **mroutes** defined by endcells A and B is called $\langle A,B\rangle$ or $\langle B,A\rangle$; see Figure 5. From now on we will only use **mroute** instead of route in both syntax and semantics.

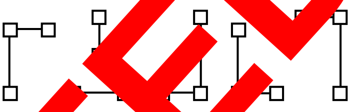


Fig. 5. mroutes

To indicate that the mroutes are congruent in terms of metric, we need the concept of *marker* as following definition.

Definition 5.1 Markers
There are three ways of representing *Markers*: (**1**) A sequence of n≥1 slash marks, or (**2**) An arc with n≥1 transversal slash marks on it, or (**3**)Line styles.

Markers will be used to represent the congruence of mroutes; see Figure 6. There are many types of markers. Two markers are of the same type iff they have the same number of slash marks, regardless of the presence or absence of the arc or the same line style.

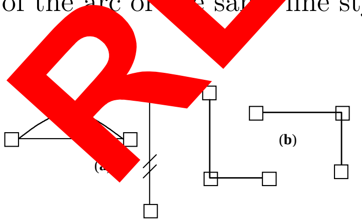


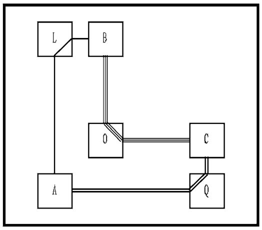
Fig. 6. Markers of same type

Therefore the two markers in (a) of Figure 6 are of the same type and those in (b) are of the same type as well. If two markers are of the same type we will just say that they are the same marker. In other words, we are only concerned with markers at the type level, not at the value level. We use α, β, γ, for markers.

Definition 5.2 Marked configuration
A *marked configuration* is a configuration in which some of the mroutes have

been *marked*. As a particular example for marking the mroutes in configuration, we can also visualise the marked mroutes by different line styles as in Figure 7, in which there are three marked mroutes $\langle B,C\rangle$, $\langle A,C\rangle$ and $\langle A,B\rangle$.

5.2 Well-Formed Configurations



H

Fig. 7. Marked configuration **H**

Every finite combination of objects is a configuration, but not all configurations are well-formed configuration (*wfc*).

Definition 5.3 Well-formed configuration
A configuration is *well-formed* iff:

(**1**) It has one and only one frame and all the other configurational objects are *in* the frame,

(**2**) Any cell must be *on* a row or column,

(**3**) There exists an mroute so that a given cell must *intersect* it,

(**4**) Every marker marks an mroute.

5.3 Configurations as Equivalence Classes

Definition 5.4 Extension of configuration
A configuration **E** is an extension of configuration **C** ($\mathbf{C} \subseteq \mathbf{E}$) iff the 1-1 function f from the set of objects of **C** to the set of objects of **E** such that:

(**1**) Configuration object x is a cell (row or column) iff $f(x)$ is a cell (row or column),

(**2**) Cell [illegible] is [illegible] row (column) y iff $f(A)$ is *on* $f(y)$,

(**3**) Cell Z *intersects* mroute $\langle A,B\rangle$ iff $f(Z)$ *intersects* mroute $\langle f(A),f(B)\rangle$, and

(**4**) If [illegible] marks $\langle A,B\rangle$ then it also marks $\langle f(A),f(B)\rangle$.

Such a [illegible] is called an *extending* of **C** into **E**.

Definition 5.5 Copy of configuration
Configuration **E** is a *copy* of configuration **C** iff f is a bijection.

Note that two configurations are copies of one another iff there is a bijection between them preserving the four relations *In*, *On*, *Intersect*, and *Marking*.

Proposition 5.6 *The copy of configuration is an equivalence relation, and for every configuration* **C**, *all the copies of* **C** *form an equivalence class.*

Proof. (Sketch) The relation of copying configuration of **C** is an equivalence relation due to meeting three *reflexive*, *symmetric* and *transitive* conditions of an equivalence one. □

From now on by **C** we will mean the equivalence class of all the configurations that are copies of **C**. If two configurations **C** and **E** are equivalent then it is denoted by **C** $\equiv$ **E**.

6 Semantics

So far, we have only talked about configurations. Now we want to know what a configuration is; i.e., what the meaning of configuration is. The meaning of configuration is expressed in the satisfaction relation ($\models$) between configuration and geometric figures in the $\mathbb{R} \times \mathbb{R}$ plane (Euclidean plane).

Configuration implies a geometric figure in the $\mathbb{R} \times \mathbb{R}$ plane
By a $\mathbb{R} \times \mathbb{R}$ plane, we mean a plane along with a finite number of points, lines (vertical and horizontal, just consider these types of lines), segments designated in lines (vertical and horizontal) such that all the points on the designated line are included among the designated points and sequences of connected segments. These elements of $\mathbb{R} \times \mathbb{R}$ planes are the *configurational objects*, as mentioned in Section 5.1, that we would like to reason about.

Geometric figure in the $\mathbb{R} \times \mathbb{R}$ plane defines a configuration
It is also easy to turn a $\mathbb{R} \times \mathbb{R}$ plane $\mathcal{P}$ into a configuration. We can do this as follows: pick any new point A in $\mathcal{P}$, pick a point B on each designated line l of $\mathcal{P}$, and let m be the maximum distance from A to any designated point, any B, or to any point on a designated line. m must be finite, since $\mathcal{P}$ only contains a finite number of designated points and lines. Let R be a circle with centre A and radius of length greater than m, and let F be a rectangle lying outside of R. Then if we let **D** be a configuration whose frame is F, whose sections are the parts of the lines (vertical and horizontal) of $\mathcal{P}$ that lie inside F, whose cells are the designated points of $\mathcal{P}$, and whose routes (mroutes) are the (two) connected line segments of $\mathcal{P}$, then **D** is a *wfc* that we call $\mathcal{P}$'s *canonical (unmarked) configuration*. Strictly speaking, we will say a canonical configuration, since the configuration we get depends on how we pick A, B and lines (vertical and horizontal); but all the configurations we can get are equivalent, so it does not really matter. We can also find $\mathcal{P}$'s *canonical marked configuration* by marking equal those mroutes in **D** that correspond to congruent segments in $\mathcal{P}$. These canonical configurations give us a convenient way of saying which $\mathbb{R} \times \mathbb{R}$ planes are represented by a given configuration.

Definition 6.1 In $\mathbb{R} \times \mathbb{R}$ plane, $\mathbf{M}$ is a *model* of the configuration **D** (in symbols, $\mathbf{M}\models$**D**, also read as '$\mathbf{M}$ satisfies **D**') if

(**1**)$\mathbf{M}$'s canonical unmarked configuration is equivalent to D's underlying unmarked configuration, and

(**2**)if two mroutes are marked equal in D, then the corresponding mroutes are marked equal in $\mathbf{M}$'s canonical marked configuration.

This definition just says that $\mathbf{M}\models\mathsf{D}$ if $\mathbf{M}$ and D have the same topology and any mroutes that are marked congruent in D really are congruent in $\mathbf{M}$.

Satisfaction relation ($\models$) is well-defined on equivalence classes of configurations

From the definitions, every $\mathbb{R}\times\mathbb{R}$ plane is the model of some configuration, namely its canonical underlying configuration, and that if D and E are equivalent configurations, then if $\mathbf{M}\models\mathsf{D}$, then $\mathbf{M}\models\mathsf{E}$. In other words, the *satisfaction* relation ($\models$) is well-defined on equivalence classes of configurations.
The full converse of this statement, that if $\mathbf{M}\models\mathsf{D}$ and $\mathbf{M}\models\mathsf{E}$, then D is equivalent to E, is not true, since D and E may have different markings. However, it is true if D and E are unmarked. Also, if D is a configuration that is not well-formed, then it has no models.

7 Proofs

Definition 7.1 Constructibility

A configuration E is said to be *constructible* from configuration D if there is a sequence of configurations beginning with D and ending with E such that each configuration in the sequence is the result of applying one of the *construction*, *inference* or *transformation* rules to the preceding configuration; such a sequence is called a *construction*. These rules will be explained in the sections below.

Sometimes one of configurations that is constructible from a configuration by a construction, inference or transformation rule cannot represent any possible situation. In that case we will say that it is semantically contradictory and we will delete it from a construction.

Definition 7.2 Contradictory configuration

Configuration D is *semantically contradictory* iff it does not have any model.

Definition 7.3 Geometric Consequence

Configuration E is a *geometric consequence* of Configuration D, and write $\mathsf{D}|\subseteq \mathsf{E}$ iff every model of D can be extended to a model of E.

Definition 7.4 Provability

Configuration E is *provable* from D, and write $\mathsf{D} \vdash \mathsf{E}$ iff there is a construction from D to E.

7.1 Construction rules

We would now like to be able to use configurations to model and encompass constructions. In order to do this, we will define several configuration construction rules. The result of applying a rule to a given *wfc* D is a configuration array of all the *wfcs*.The configuration construction rules are given below:

Rules for frame:

C0.1. A frame must be added if it does not already exist.

Rules for cell:

C1.1. A cell may be added to the interior of frame or on any existing row (column).

C1.2. A row and column *intersect* at a cell.

Rules for row(column):

C2.1. A row (column) may be added to the interior of frame.

C2.2. A row (column) may be added to go *through* any existing cell.

Rules for section and semi-section:

C3.1. A section may be added to any two given existing distinct adjacency cells *on* row (column) if there is not already one existing.

C3.2. Any section can be extended to a full row (column).

C3.3. A semi-section may be added to any side of frame border and a given existing cell, *on* row (column), being adjacent to that border side if there is not already one existing.

Rules for mroute:

C4.1. Given two distinct cells A and B, an mroute may be added whose endcells are A and B.

Rules for marker:

C5.1. Every mroute may be marked with a marker if there is not already one existing, and any mroutes of two same endcells must be marked with the same marker.

Rules for deleting an object

C6.1. Any row (column) or mroute may be erased; any section or semi-section of a row (column) may be erased; and any cell that is not an *intersection* of mroute or of one row and column and is not *on* a section may be erased. If an mroute is erased, any marking that marks it must also be erased.

Rules for array

C7.1. Any new configuration can be added to a given configuration array.

Note that rule **C3.1** is a special case of rule **C4.1**, while **C4.1** is derivable from **C3.1**, as defined above in section of syntax.

Example: Applying some construction rules for setting up a particular configuration

Let us consider the configurations shown in Figure 8. What happens if we apply some construction rules to create the possible configurations including three cells A, B and C and mroutes connecting them ?

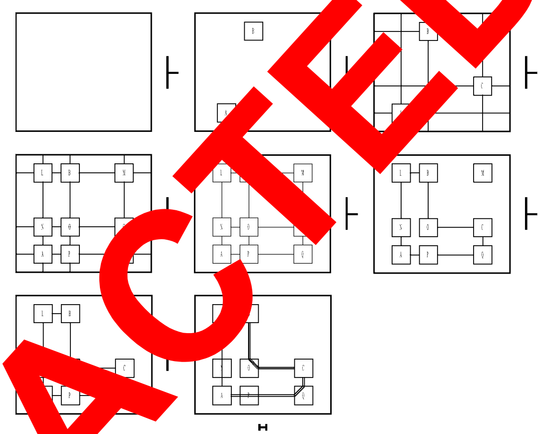


Fig. 8. An example for applying some construction rules to create a well-formed configuration

$\vdash$**A** by rule **C0.1**.**A**$\vdash$**B** (Applying rule **C1.1** to create three cells A, B and C). **B**$\vdash$**C** (Applying rule **C2.2** to create three rows and three columns go through A, B, C). **C**$\vdash$**D** (Applying rule **C1.2** to create the cells at the intersections of the rows and columns). **D**$\vdash$**E** (Applying rule **C6.1** to delete all semi-sections). **E**$\vdash$**F** (Applying rule **C6.1** to delete four sections $[B,M]$, $[M,C]$, $[N,C]$ and $[P,C]$).**F**$\vdash$**G** (Applying rule **C6.1** to delete the cell M). **G**$\vdash$**H** (Applying rule [illegible] to create three mroutes $\langle A,B\rangle$ for the two cells A, B, $\langle B,C\rangle$ for the two cells B, C and $\langle A,C\rangle$ for the two cells A, C, and rule **C**[illegible] to create three markers on these three mroutes).

7.2 Inference rules

Once we have constructed a configuration, we would like to be able to reason about it. For this purpose, we have rules of inference.

Rules for marker:

R1.1. Any mroute may be marked with a new marker. Any marker can be removed from any configuration.

Rules for transitivity of marking:

R2.1. If two mroutes a and b are marked with the same marker and, in addition, a is also marked with another new marker, then b may also be marked with that same new marker.

R2.2. If any two cells on two distinct rows and columns, there exists two rmoutes that connect them, then their component sections on rows (columns) are congruent to each other.

R2.3. Any two mroutes are congruent if their respective component sections are equivalent.

Rules for reduction:

R3.1. Given a configuration array that contains two identical configurations, one of them may be removed.

R3.2. If any configuration contains a two-sections mroute, and such mroute and any one of its sections are marked with the same marker, then it can be removed from a configuration array.

7.3 Transformation rules

We would also like to be able to use configurations to model isometries: *translations*, *rotations*, and *reflections*. To do this we first need the notion of a *subconfiguration* and *super transformation configuration*.

Definition 7.5 Subconfiguration
A configuration **A** is a *subconfiguration* of **B** if **A** is constructible from **B** using only rule **C6.1**.

Definition 7.6 Unreversed and reversed equivalence
A and **B** are two *unreversed equivalent* configurations (or equivalent for short) if mroutes of **A** traverse clockwise (counter-clockwise), then corresponding ones of **B** also clockwise (counter-clockwise). In other side, **A** and **B** are two *reversed* equivalent configurations if mroutes of **A** traverse clockwise (counter-clockwise), then corresponding ones of **B** counter-clockwise (clockwise).

Definition 7.7 Super transformation configuration
A configuration **T** is an *super transformation configuration* of **A** (via transformation t) if **A** is a subconfiguration of **T**, and there exists another configuration **B** and a function t: **A** $\rightarrow$ **B** such that **B** is also a subconfiguration of **T**, and **A** and **B** are equivalent or reverse equivalent configurations via the map t.

Definition 7.8 Transformation configuration
T is a transformation configuration of **A** via t if **T** is a super transformation configuration of **A** via t, and there is no configuration **S** such that **S** is constructible from **T** by rule **C6.1** and **S** is still a transformation configuration of **A** via t.

Definition 7.9 Unreversed and reversed transformation configuration
If **A** and **B** are equivalent, then it is an *unreversed* transformation configura-

tion, and if they are reverse equivalent, then it is a *reversed* transformation configuration.

Now we can incorporate symmetry transformations into our computing system by adding the rules as below.

Rules for gliding:

S1.1. Given a configuration **D**, the subconfiguration **C**, a cell A and a section l ending at A in **C**, and a cell B and a section m ending at B in **D**, the result of applying this rule is the configuration array of all unreversed transformation configurations of **C** in **D** such that $t(A) = B$ and $t(l)$ lies along the same row (column) as m, on the same side of B as m.

Rules for reflected gliding:

S2.1. Given a configuration **D**, the subconfiguration **C**, a cell A and a section l ending at A in **C**, and a cell B and a section m ending at B in **D**, the result of applying this rule is the configuration array of all reversed transformation configurations of **C** in **D** such that $t(A) = B$ and $t(l)$ lies along the same row (column) as m, on the same side of B as m.

Note that simple translations and rotations are special cases of rule **S1.1**, and reflections are a special case of rule **S2.1**.

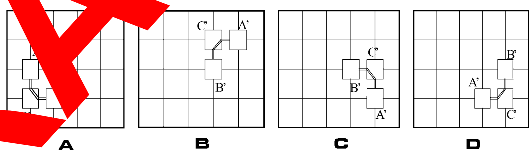


Fig. 9. **A** $\vdash$ {**B**,**C**,**D**} by rule **S1.1**

Example: Applying the gliding rule

As a relatively simple example of how these rules work, consider the configuration **A** shown in Figure 9. By rule **S1.1** we will obtain a configuration array including the configurations **B**, **C**, and **D** as below.

7 Soundness of construction, inference and transformation rules

A construction, inference and transformation rule is said to be *sound* if it always models a possible real construction, meaning that if **M** $\models$ **D** and configuration **E** follows from **D** via this rule, then **M** can be extended to a model of **E**. The rules given as above are sound, because in any model, we can add new points, connect two points on a line (horizontal, vertical) by a segment, extend any segment to a line (horizontal, vertical), or draw a sequence of segments connecting any point with a given point, and we can erase points, segments, and lines (horizontal, vertical). Moreover these elements also meet all deductive principles of inference and transformation in geometry. In general, if every model $\mathbf{M}$ of D can be extended to a model of E, then as mentioned definition above we say that E is a *geometric consequence of* D, and write $\mathsf{D}|\!\subset\mathsf{E}$.

Theorem 7.10 soundness: *If configuration* E *is* provable *from configuration* D ($\mathsf{D}\vdash\mathsf{E}$) *then* $\mathsf{D}|\!\subset\mathsf{E}$.

Proof. It is trivial because the construction, inference and transformation rules are sound; it follows by induction on the length of constructions that if E is provable from D, then E is a geometric consequence of D. □

8 Summary

We have defined a clear abstract syntax for some particular types of objects satisfying certain conditions and their algebraic notations for representing a logical configuration. We divided the configurational objects into two different classes, namely primitive and derived classes. The primitive objects are not defined. The derived objects are defined in terms of the primitive objects. The formal semantics was seen as arrangements of objects in a given logical configuration and as the satisfaction relation between configurations and geometric figures in the Euclidean plane. Also, we formulated rules of construction, transformation and inference, and then proved them to be sound. We have taken a methodological stance. Additional work is required to further formalize some of the concepts we have introduced here. We are currently engaged in this activity and expect to report on this more fully in the future.

References

[1] G. Allwein and J.Barwise, *Logical Reasoning with Diagrams*, Oxford, England: Oxford University Press, 1996.

[2] R. Bardohl and M. Minas and A. Schurr and G. Taentzer, *Application of graph transformation to visual languages*, In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, Handbook of Graph Grammars and Computing by Graph Transformation, volume II: Applications, Languages and Tools, World Scientific, 1999, pp. 105–180.

[3] W.S.Carter, K.Duong, R.H.Freeman, H.C.Hsieh, J.Y.ja, J.E.Mahoney, L.T.Ngo and S.L.Sze, *A User Programmable Reconfigurable Gate Array.* IEEE Custom Integrated Circuits Conference, 1986, pp. 233-235.

[4] N.G.Miller, *A Diagrammatic Formal System for Euclidean Geometry*, PhD thesis, Cornell University, US,2001.

[5] E.C.Wallace and S.F.West, *Roads to Geometry*, Prentice Hall, 1992.