# Expressing Properties in Second and Third Order Logic: Hypercube Graphs and SATQBF [1]

FLAVIO FERRAROTTI, *School of Information Management, Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand. E-mail: flavio.ferrarotti@vuw.ac.nz*

WEI REN, *School of Engineering and Advanced Technology, Massey University, Private Box 756, Wellington 6140, New Zealand. E-mail: w.ren@massey.ac.nz*

JOSE MARIA TURULL TORRES, *ICTIC, Universidad de la Cuenca del Plata, Corrientes, Argentina and Department of Informatics, Universidad Nacional de San Luis, Ejercito de Los Andes 950, D5700HHW, San Luis, Argentina. E-mail: J.M.Turull@massey.ac.nz*

## Abstract

It follows from the famous Fagin's theorem that all problems in NP are expressible in existential second-order logic (∃SO), and vice versa. Indeed, there are well-known ∃SO characterizations of NP-complete problems such as 3-colorability, Hamiltonicity and clique. Furthermore, the ∃SO sentences that characterize those problems are simple and elegant. However, there are also NP problems that do not seem to possess equally simple and elegant ∃SO characterizations. In this work, we are mainly interested in this latter class of problems. In particular, we characterize in second-order logic the class of hypercube graphs and the classes $\mathrm{SATQBF}_k$ of satisfiable quantified Boolean formulae with $k$ alternations of quantifiers. We also provide detailed descriptions of the strategies followed to obtain the corresponding nontrivial second-order sentences. Finally, we sketch a third-order logic sentence that defines the class $\mathrm{SATQBF} = \bigcup_{k\geq 1}\mathrm{SATQBF}_k$. The sub-formulae used in the construction of these complex second- and third-order logic sentences, are good candidates to form part of a library of formulae. Same as libraries of frequently used functions simplify the writing of complex computer programs, a library of formulae could potentially simplify the writing of complex second- and third-order queries, minimizing the probability of error.

*Keywords*: second-order logic, third-order logic, quantified Boolean formulae, queries, finite model theory, hypercube graphs

## 1 Introduction

Examples of second-order formulae expressing different properties of graphs are fairly common in the literature. Classical examples are 3-*colorability*, *Hamiltonicity*, and *clique* (see [8, 10] among others). These properties can be expressed by simple and

[1] Pre-print of article submitted to an special issue of the Logic Journal of the IGPL with selected papers from the 16th Brazilian Logic Conference.

 

elegant second-order formulae. Likewise, there are graph properties that can be expressed by simple and elegant third-order formulae. One of those properties is that of being a *hypercube graph* (see [5]). An *n-hypercube graph* $\mathbf{Q}_n$, also called an $n$-cube, is an undirected graph whose vertices are binary $n$-tuples. Two vertices of $\mathbf{Q}_n$ are adjacent iff they differ in exactly one bit.

The expressive power of third-order logic is not actually required to characterize hypercube graphs, since they can be recognized in nondeterministic polynomial time. Recall that by Fagin's theorem [4], $\exists$SO captures NP. Thus there are formulae in existential second-order logic ($\exists$SO) which can express this property. Nevertheless, to define the class of hypercube graphs in second-order logic is certainly more challenging than to define it in third-order logic.

From an applied perspective, this indicates that it makes sense to investigate higher-order quantifiers in the context of database query languages. Despite the fact that most of the queries commonly used in the industry are in P, the use of higher-order quantifiers can potentially simplify the way in which many of those queries are expressed.

Let SATQBF$_k$ denote the class of satisfiable quantified Boolean formulae with $k$ alternating blocks of quantifiers. From Fagin-Stockmeyer characterization of the polynomial-time hierarchy [13] and the fact that SATQBF$_k$ is complete for the level $\Sigma_k^p$ of that hierarchy [14], it follows that for every $k \geq 1$, SATQBF$_k$ can be defined by a formula in the prenex fragment $\Sigma_k^1$ of second-order logic with $k$ alternating blocks of quantifiers. SATQBF$_k$ provides a prime example of a property (or query) whose expression in the language of second-order logic is possible but challenging. Indeed, it is not a trivial task to write a second-order logic sentence that evaluates to true precisely on those word models that represent sentences in SATQBF$_k$. As usual in finite model theory [3], the term word model refers here to a finite relational structure formed by a binary relation and a finite number of unary relations. By contrast, if we restrict our attention to quantified Boolean formulae in which the quantified free part is in conjunctive normal form and has exactly three Boolean variables in each conjunct, then the problem is expressible in monadic second-order logic provided that the formulae are encoded using a different kind of finite relational structures which include ternary relations (see [10]).

Thus, on the one hand there are well-known NP-complete problems such as 3-colorability, Hamiltonicity and clique, that have corresponding well-known characterizations in $\exists$SO which are simple and elegant. Those characterizations have in common that the existential second-order quantifiers can be identified with the guessing stage of the NP algorithm, and that the remaining first-order formula corresponds to the polynomial time deterministic verification stage. On the other hand, there are well-known problems such as hypercube graph (which can also be characterized in $\exists$SO) and SATQBF$_k$ (which can be characterized in $\Sigma_k^1$) that do not appear to have a straightforward characterization in second-order logic, even if we consider the full second-order language.

This observation prompted us to write second-order characterizations of hypercube graph and SATQBF$_k$. The resulting second-order sentence for hypercube graph can be found in [11]. The corresponding sentence for SATQBF$_k$ was included in [12]. Both sentences are complex and several pages long. In this article we present a detailed description of the strategies followed to write these sentences. The sub-formulae used

for the implementation of these strategies could be part of a future library of second-order formulae. Same as libraries of frequently used functions simplify the writing of complex computer programs, a library of formulae could potentially simplify the writing of complex second-order queries, minimizing the probability of error.

The minimization of the probability of error constitutes an important objective in the context of this work, since given a query $q$ and a second-order formula $\varphi$, it is *not* possible to formally prove whether $\varphi$ expresses $q$. For this reason, we make use of full second-order logic to present the characterizations of hypercube graph and $\text{SATQBF}_k$, even though its $\exists\text{SO}$ and $\Sigma^1_k$ fragments, respectively, already have the expressive power required for these tasks. This has permitted us to write relatively clear and intuitive formulae as well as to follow a top-down strategy, similar to that commonly used in the development of computer programs, to further reduce the chance of error.

If we consider the whole class $\text{SATQBF} = \bigcup_{k\geq 1}\text{SATQBF}_k$ of satisfiable quantified Boolean formulae, then the problem becomes PSPACE-complete. Since PSPACE can be captured by second-order logic extended with a transitive closure operator, and furthermore this logic is widely conjectured to be strictly more expressive than the standard second-order logic, the existence of a second-order logic characterization of this problem is unlikely. Thus, we decided to look for a characterization in third-order logic. Note that it is a well-known fact that third-order logic is powerful enough as to characterize every problem in PSPACE. We conclude the paper presenting a sketch of a third-order logic sentence that defines the class SATQBF. That is, we present a strategy to write a third-order sentence that evaluates to true precisely on those word models that represent sentences in SATQBF.

We strongly believe that in many respects the descriptive approach to Complexity is more convenient than the classical one. That is, using formulae of some logic to study upper bounds in the time or space complexity of a given problem, instead of Turing machines. There are many different measures which can be taken on the formulae that express a given problem such as quantifier rank, quantifier blocks alternation, number of variables, number of binary connectives, and arity of quantified relation variables. It has been proved that bounds on those measures impact on the expressive power of logics over finite models (see [10], [3], [8]). Furthermore, it is rather obvious that all those measures are decidable, in contrast to the use of Turing machines, where the usual measures relevant to computation power such as time, space, treesize, and number of alternations, are clearly undecidable. Regarding lower bounds there are also several well studied and powerful techniques in Descriptive Complexity which proved to be extremely useful in the last decades, such as Ehrenfeucht-Fraisse games and their variations (see [9] in particular) and 0-1 Laws (again see [10], [3], [8]).

Hence, it is important to learn how to build formulae which are large, but still intuitive and clearly understandable in a top down approach, in the same way that this is important in the construction of algorithms in the classical approach to Complexity, which are also clear and intuitive no matter their size. The work reported in this article is to the authors' knowledge one of the first steps in that direction.

In the next section, we introduce the necessary notation and formally describe by means of a third-order logic sentence, the class of hypercube graphs. In Section 3, we define in second-order logic the basic arithmetic operations that we need for this work. We describe the strategy used to characterize the class of hypercube graphs in the language of second-order logic in Section 4. In Section 5 we formally describe

the problems SATQBF$_k$ and SATQBF, and we consider their complexity. In Section 6, we explain in full detail how to build for each $k \geq 1$, a second-order sentence that expresses $SATQBF_k$. In Section 7 we explain how to build a third-order logic sentence which expresses SATQBF, and we give a sketch of such formula. Finally in Section 8, we present some final considerations.

## 2 Background

We assume that the reader is acquainted with the basic concepts and the framework of Finite Model Theory [3, 10]. We use the notation from [10].

We work on the vocabulary $\sigma = \{E\}$ of graphs. An *undirected graph* $\mathbf{G}$ is a finite relational structure of vocabulary $\sigma$ satisfying $\varphi_1 \equiv \forall xy(E(x,y) \rightarrow E(y,x))$ and $\varphi_2 \equiv \forall x(\neg E(x,x))$. If we do not require $\mathbf{G}$ to satisfy neither $\varphi_1$ nor $\varphi_2$, then we speak of a *directed graph* (or *digraph*). We denote as $V$ the domain of the structure $\mathbf{G}$, i.e., the set of vertices of the graph $\mathbf{G}$. The edge relation of $\mathbf{G}$ is denoted as $E^{\mathbf{G}}$.

By *second-order logic* we refer to the logic that is obtained when first-order logic is extended with second-order variables which range over subsets and relations defined over the domain, and quantification over such variables. As usual, we use uppercase letters $X, Y, Z, \ldots$ to denote second-order variables and lower case letters $x, y, z, \ldots$ to denote first-order variables. The arity of the second-order variables that we use in our formulae is always clear from the context. See [10] or [3] for a formal definition of second-order logic in the context of finite model theory. We include an example of a second-order formula that defines a simple graph property instead.

Example 2.1
An undirected graph $\mathbf{G}$ is *regular* if all its vertices have the same degree. It is well known that the class of regular graphs is not definable in first-order logic [3, 8]. In second-order logic, this class can be defined as follows:

$\exists A\big(\forall x\big(\exists B(\text{A1} \wedge \text{A2})\big)\big)$ where

- A1 expresses "$B$ is the set of vertices which are adjacent to $x$".
  A1 $\equiv \forall z\big(B(z) \leftrightarrow E(x,z)\big)$
- A2 expresses "the sets $A$ and $B$ have the same cardinality" with a formula stating that there is a bijection $F$ from $A$ to $B$.
  A2 $\equiv \exists F \forall xyz\big(\text{A2.1} \wedge \text{A2.2} \wedge \text{A2.3} \wedge \text{A2.4} \wedge \text{A2.5}\big)$ where
  – A2.1 means "$F$ is a subset of $A \times B$".
    A2.1 $\equiv \big(F(x,y) \rightarrow A(x) \wedge B(y)\big)$
  – A2.2 means "$F$ is a function".
    A2.2 $\equiv \big(F(x,y) \wedge F(x,z) \rightarrow y = z\big)$
  – A2.3 means "$F$ is total".
    A2.3 $\equiv \big(A(x) \rightarrow \exists y(F(x,y))\big)$
  – A2.4 means "$F$ is injective".
    A2.4 $\equiv \big(F(x,z) \wedge F(y,z) \rightarrow x = y\big)$
  – A2.5 means "$F$ is surjective".
    A2.5 $\equiv \big(B(y) \rightarrow \exists x(F(x,y))\big)$

We say that a sentence $\varphi$ expresses a Boolean query $q$ (or property) over finite relational structures of vocabulary $\sigma$, if for every finite relational structure $\mathbf{G}$ of

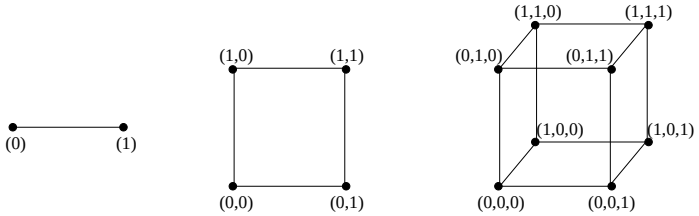


Figure 2.1

vocabulary $\sigma$, $q(\mathbf{G})$ = true iff $\mathbf{G} \models \varphi$. For instance the sentence in Example 2.1 expresses the Boolean query: Is $\mathbf{G}$ a regular graph? We denote by $Mod(\varphi)$ the class of finite $\sigma$-structures $\mathbf{G}$ such that $\mathbf{G} \models \varphi$. A class of finite $\sigma$-structures $\mathcal{C}$ is *definable* in a logic $\mathcal{L}$, if $\mathcal{C} = Mod(\varphi)$ for some $\mathcal{L}$-sentence $\varphi$ of vocabulary $\sigma$. For instance the class of regular graphs is definable in second-order logic, as shown by the formula given in Example 2.1.

Next, we define the class of hypercube graphs using a relatively simple and elegant formula in *third-order logic*. This logic extends second-order logic with third-order variables which range over subsets and relations defined over the powerset of the domain, and quantification over such variables. We use uppercase calligraphic letters $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \ldots$ to denote third-order variables. A formal definition of higher-order logics in the context of finite model theory can be found in [7] among others.

Example 2.2
An *n-hypercube* (or *n-cube* for short) $\mathbf{Q}_n$ can be defined as an undirected graph whose vertices are all the binary $n$-tuples. Two vertices of $\mathbf{Q}_n$ are adjacent iff they differ in exactly one bit. A 1-cube $\mathbf{Q}_1$, a 2-cube $\mathbf{Q}_2$ and a 3-cube $\mathbf{Q}_3$ are displayed in Figure 2.1.

We can build an $(n+1)$-cube $\mathbf{Q}_{n+1}$ starting with two isomorphic copies of an $n$-cube $\mathbf{Q}_n$ and adding edges between corresponding vertices. Using this fact, we can define in third-order logic the so called class of *hypercube graphs*, as follows:

$\exists\mathcal{C}\exists\mathcal{O}\big(\text{A1} \wedge \text{A2} \wedge \forall G_1 \forall G_2\big((\mathcal{C}(G_1) \wedge \mathcal{C}(G_2) \wedge \text{A3}) \rightarrow \text{A4}\big) \wedge \text{A5} \wedge \text{A6}\big)$ where

- A1 expresses "$\mathcal{C}$ is a class of undirected graphs".
- A2 expresses "$\mathcal{O}$ is a total order on $\mathcal{C}$".
- A3 expresses "$G_1$ is the immediate predecessor of $G_2$ in the order $\mathcal{O}$".
- A4 expresses "$G_2$ can be built from two isomorphic copies of $G_1$ by adding edges between the corresponding vertices".
- A5 expresses "the first graph in the order $\mathcal{O}$ is a $Q_1$".
- A6 expresses "the last graph in the order $\mathcal{O}$ is the input graph".

In turn, we can express A4 as follows:

$\exists F_1 \exists F_2\big(\text{A4.1} \wedge \text{A4.2} \wedge \text{A4.3} \wedge \forall x(x \in dom(G_1) \rightarrow \text{A4.4}) \wedge$
$\neg\exists xy(x, y \in dom(G_1) \wedge x \neq y \wedge \text{A4.5})\big)$ where

- A4.1 expresses "$F_1$ and $F_2$ are injective and total functions from $dom(G_1)$ to $dom(G_2)$".

- A4.2 expresses "the ranges of $F_1$ and $F_2$ form a partition of $dom(G_2)$".
- A4.3 expresses "$F_1$ and $F_2$ are isomorphisms from $G_1$ to the sub-graphs of $G_2$ induced by the ranges of $F_1$ and $F_2$, respectively".
- A4.4 expresses "there is an edge in $G_2$ which connects $F_1(x)$ and $F_2(x)$".
- A4.5 expresses "there is an edge in $G_2$ which connects $F_1(x)$ and $F_2(y)$".

Note that, if there is an edge $(a, b)$ in $G_2$ such that $a$ belongs to the range of $F_1$ and $b$ belongs to the range of $F_2$, or vice versa, then either $F_1^{-1}(a) = F_2^{-1}(b)$ or $F_1^{-1}(b) = F_2^{-1}(a)$.

The missing logic formulae in this example are left as an exercise for the reader.

The property of a graph being an $n$-cube for some $n$, is known to be in NP. A nondeterministic Turing machine can decide in polynomial time whether an input structure $\mathbf{G}$ of the vocabulary $\sigma$ of graphs is an hypercube, by simply computing the following steps:

i. Compute the logarithm in base 2 of the size $n$ of the domain of the input structure $\mathbf{G}$ which must be a positive integer;
ii. Guess a sequence $s_1, \ldots, s_n$ of $n$ binary strings, each of length $\log_2 n$;
iii. Check in polynomial time that all binary strings are unique, that the sequence contains all binary strings of length $\log_2 n$ and that, for some ordering $a_{s_1}, \ldots, a_{s_n}$ of the nodes in $V$, a string $s_i$ differs from a string $s_j$ in exactly 1 bit iff there is an edge $(a_{s_i}, a_{s_j}) \in E^{\mathbf{G}}$.

Thus, as we mentioned in the introduction, the full expressive power of third-order logic is not actually needed to characterize the class of hypercube graphs. In fact, there is a formula in ∃SO which can express this property. Recall that by Fagin's theorem [4], ∃SO captures $NP$. However, it is very unlikely that there is a formula in second-order logic, not to mention in ∃SO, that expresses the property in a way which is as intuitive and simple as in the example above.

## 3 Arithmetic in Second-Order Logic

We define in this section the basic arithmetic operations of addition, multiplication and exponentiation in second-order logic over finite structures. We encode initial segments of natural number as finite relational structures by using linear digraphs. Let $\mathbf{G}$ be a linear digraph. The first (root) element of the domain in the order determined by the edge relation $E^{\mathbf{G}}$ represents the 0, the second element in this order represents the 1, the third element represents the 2 and so on. Since in a linear digraph, $E^{\mathbf{G}}$ is the successor relation, for clarity we use $\mathrm{succ}(x, y)$ to denote $E(x, y)$. We also use $x = n$ where $n > 0$ to denote the formula of the form

$$\exists y(\mathrm{succ}(y, x) \wedge \exists x(\mathrm{succ}(x, y) \wedge \exists y(\mathrm{succ}(y, x) \wedge \cdots \wedge \varphi) \cdots))$$

with $n$ nested quantifiers and $\varphi \equiv \neg\exists x(\mathrm{succ}(x, y))$ if $n$ is odd or $\varphi \equiv \neg\exists y(\mathrm{succ}(y, x))$ if $n$ is even. Likewise, $x = 0$ denotes $\neg\exists y(\mathrm{succ}(y, x))$. We assume a total order $\leq$ of the nodes in $V$ such that $x \leq y$ iff there is a path from $x$ to $y$ in $\mathbf{G}$ or $x = y$. This total order is easily definable in second-order logic.

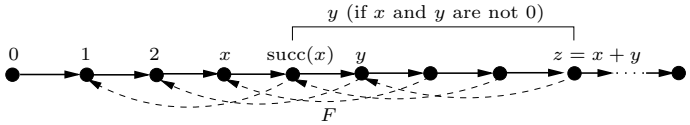


Figure 3.1: Addition

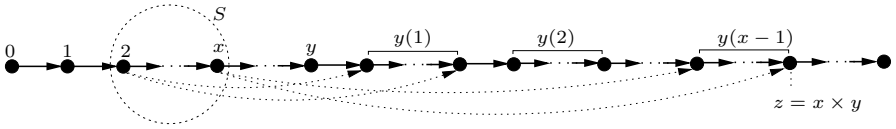


Figure 3.2: Multiplication

Let us start by defining the operation of *addition*. The strategy is depicted in Figure 3.1 in which we show the result $z$ of adding $x$ and $y$ along a linear graph.

The predicate $\mathrm{sum}(x, y, z)$, which is true iff $z = x + y$, can be defined in second-order logic as follows.

$$\big(x = 0 \wedge z = y\big) \vee \big(y = 0 \wedge z = x\big) \vee$$
$$\big(x \neq 0 \wedge y \neq 0 \wedge \exists F\big(\mathrm{A1} \wedge F(z, y) \wedge \exists x' y'(\ \mathrm{succ}(x, x') \wedge F(x', y') \wedge y' = 1) \wedge$$
$$\forall x' y' x'' y''((\ \mathrm{succ}(x', y') \wedge F(x', x'') \wedge F(y', y'')) \rightarrow\ \mathrm{succ}(x'', y''))\big)\big)$$

where A1 expresses "$F$ is an injective function with domain $\{n \in V \mid \mathrm{succ}(x) \leq n \leq z\}$". It is an easy and supplementary task to write the actual formula corresponding to A1. For the sake of clarity, we avoid this kind of supplementary details from now on.

The next arithmetic operation that we define is *multiplication*. The strategy is depicted in Figure 3.2 in which we show the result $z$ of $x$ times $y$. Each of the nodes in the subset $S = \{2, \ldots, x\}$ can be considered as a root of a different ordered tree in a forest. Each root node in the forest has $y$ children and the result $z$ is the last child of node $x$.

The predicate $\mathrm{times}(x, y, z)$, which is true if $z = x \times y$, can be defined in second-order logic as follows.

$$(x = 1 \wedge y \neq 0 \wedge z = y) \vee (y = 1 \wedge x \neq 0 \wedge z = x) \vee ((x = 0 \vee y = 0) \wedge z = 0) \vee$$
$$\big(x \neq 0 \wedge y \neq 0 \wedge x \neq 1 \wedge y \neq 1 \wedge$$
$$\exists S\big(\forall u\big(\big((2 \leq u \wedge u \leq x) \rightarrow [\exists y'(S(u, y')) \wedge$$
$$\forall x' y'((x' \leq y' \wedge x' \neq y' \wedge S(u, x') \wedge S(u, y')) \rightarrow$$
$$\neg\exists z'(x' \leq z' \wedge z' \leq y' \wedge \neg S(u, z'))) \wedge$$
$$\exists F(\mathrm{A1}) \wedge \mathrm{A2} \wedge \mathrm{A3} \wedge \mathrm{A4}]\big) \wedge$$
$$\mathrm{A5} \wedge \forall uv(S(u, v) \rightarrow (2 \leq u \wedge u \leq x))\big)\big)\big) \text{ where}$$

- A1 expresses "$F$ is a bijection from $\{n \in V \mid S(u, n)\}$ to $\{n \in V \mid 1 \leq n \leq y\}$, which means that the output degree of $u$ is $y$".
- A2 expresses "if $u = 2$ then the first child of $u$ is $\mathrm{succ}(y)$".

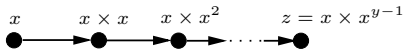


Figure 3.3: Exponentiation

- A3 expresses "if $u = x$ then the last child of $u$ is $z$".
- A4 expresses "if $u \neq 2$ then $\text{succ}(c_{u-1}, c_u)$ for $c_{u-1}$ the last child of $u-1$ and $c_u$ the first child of $u$".
- A5 expresses "the input degree of every node in $S$ is $\leq 1$".

Finally, we need to define the arithmetic operation of *exponentiation* in second-order logic. In this case, the strategy is depicted in Figure 3.3. Note that, the first node in the linear digraph is $x^1$, the second node is $x^2$, and so on till node $y$-th (the final node) which is $x^y$.

The predicate $\exp(x, y, z)$, which is true if $z = x^y$, can be defined in second-order logic as follows.

$$(x \neq 0 \wedge y = 0 \wedge z = 1) \vee (y = 1 \wedge z = x) \vee (x = 1 \wedge z = 1) \vee$$
$$\big(x \geq 2 \wedge y \geq 2 \wedge \exists V' E' \big(\text{A1} \wedge \exists F(\text{A2}) \wedge$$
$$\forall u(\neg V'(u) \vee (u = x \vee \exists x'(E'(x', u) \wedge \text{times}(x, x', u))))\big)\big) \text{ where}$$

- A1 expresses "$(V', E')$ is a linear digraph whose first (root) node is $x$ and whose last (leaf) node is $z$".
- A2 expresses "$F$ is a bijection from $V'$ to $\{1, \ldots, y\}$, i.e., $|V'| = y$".

## 4 Hypercube Graph in Second-Order Logic

We describe in this section two different strategies to define in second-order logic the class of hypercube graphs. The first strategy is based in the usual definition of Hypercube graph which identifies the nodes of the graph with binary strings. This definition was explained and expressed by means of a third-order logic formula in Example 2.2. The second strategy is based in the following definition: An $n$-hypercube graph is a graph with $2^n$ nodes, which correspond to the subsets of a set with $n$ elements. Two nodes labelled by subsets $S_i$ and $S_j$ are joined by an edge if and only if $S_i$ can be obtained from $S_j$ by adding or removing a single element. The first strategy resulted in a more cumbersome formula than the formula produced by the second strategy. However, the descriptive complexity of the formula produced by this latter strategy is higher.

### *4.1 First Strategy*

The idea is to use binary encodings to represent each node in the graph, and then to compare the binary encodings of two connected nodes to identify whether they differ exactly in 1 bit. Following a top down approach to the problem, we start with a very general schema of the formula and then we explore the main sub-formulae involved in the solution. We aim for a good balance between level of detail and

clarity of presentation. Consequently, we leave out of the presentation some trivial sub-formulae which are not central to the general strategy.

Let $\mathbf{G}$ be an undirected graph with $|V| = n$. The following second-order formula is satisfied by $\mathbf{G}$ iff $\mathbf{G}$ is an *m-hypercube graph* for some $m$.

$$\varphi_1 \equiv \exists \leq \big(\text{A1} \wedge \exists F\, \exists m\big(\text{A2} \wedge \forall xy(E(x,y) \leftrightarrow \text{A3}) \wedge \text{A4}\big)\big) \text{ where}$$

- A1 expresses "$\leq$ is a total order of the domain $V$ of $\mathbf{G}$".
- A2 expresses "$F$ is a bijection on $V$".
- A3 expresses "The binary encodings of $F(x)$ and $F(y)$ have both length $m$ and differ exactly in one bit".
- A4 expresses "There is a node whose binary encoding contains no zeros".

The total order $\leq$ is used to identify each individual node of $V$. Thus, we can assume that $V = \{0, \ldots, n-1\}$. This is needed for the binary encoding of the nodes in $V$, as it will become clear latter on. It should be clear how to express A1 and A2 in the language of second-order logic. Thus we concentrate our effort in explaining the strategy to express A3. Finally, note that A4 means that all binary encodings (of length $m$) correspond to some node in $V$, which implies that the number of nodes of $\mathbf{G}$ is a power of 2, and also that $m = \log_2 n$. A sub-formula that expresses A4 can be easily built by using the same ideas that we use for A3 below. That is, we can existentially quantify for some node $z$, a linear digraph $(V_z, E_z)$ and a Boolean assignment $B_z$ which assigns 1 to each node, and such that the binary string represented by $(V_z, E_z, B_z)$ is the binary encoding of $F(z)$.

The following formula expresses A3.

$$\begin{aligned}
&\exists V_x E_x V_y E_y B_x B_y \big(\text{A3.1} \wedge \text{A3.2} \wedge \text{A3.3} \wedge \text{A3.4} \wedge \text{A3.5} \wedge \\
&\qquad \exists G\big(\ \text{A3.6} \wedge \\
&\qquad\quad \forall uv((E_x(u,v) \rightarrow \exists u'v'(G(u,u') \wedge G(v,v') \wedge E_y(u',v'))) \wedge \\
&\qquad\qquad (E_y(u,v) \rightarrow \exists u'v'(G(u',u) \wedge G(v',v) \wedge E_x(u',v')))) \wedge \\
&\qquad\quad \exists v \forall v'((\text{A3.7} \rightarrow v' \neq v) \wedge \\
&\qquad\qquad\quad (\text{A3.8} \rightarrow v' = v))\big)\big) \text{ where}
\end{aligned}$$

- A3.1 expresses "$(V_x, E_x)$ and $(V_y, E_y)$ are linear digraphs".
- A3.2 expresses "$B_x$ is a function from $V_x$ to $\{0,1\}$".
- A3.3 expresses "$B_y$ is a function from $V_y$ to $\{0,1\}$".
- A3.4 expresses "$(V_x, E_x, B_x)$ is the binary encoding of $F(x)$".
- A3.5 expresses "$(V_y, E_y, B_y)$ is the binary encoding of $F(y)$".
- A3.6 expresses "$G$ is a bijection from $V_x$ to $V_y$".
- A3.7 expresses "$B_x(v') = B_y(G(v'))$".
- A3.8 expresses "$B_x(v') \neq B_y(G(v'))$".

To complete the picture, we need to explain how to write A3.4 and A3.5 in the language of second-order logic. Since both can be expressed in second-order logic in a similar way, we only show the formula for A3.4. Let $x_i$ be the $i$-th node in the

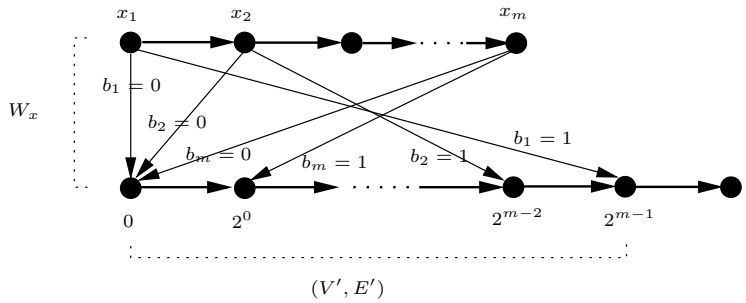


Figure 4.1

linear graph $(V_x, E_x)$ defined in the previous formula. We say that $(V_x, E_x, B_x)$ is the binary encoding of $F(x)$ if

$$F(x) = b_1 \times 2^{m-1} + b_2 \times 2^{m-2} + \cdots + b_m \times 2^0, \text{ where } b_i = B_x(x_i).$$

In second-order logic, we use a function $W_x$ which assigns to each node $x_i$ in $V_x$ its corresponding value $b_i \times 2^{m-i}$ in the encoding. This function is depicted in Figure 4.1. The following formula defines the encoding.

$\exists W_x\, I_x\, n_x\, v\, w\, \forall x' \big($A3.4.1$\wedge$
$\quad \forall s\, s'\, q\, q'((E_x(s,q) \wedge I_x(s,s') \wedge I_x(q,q')) \rightarrow \text{succ}(s',q'))\wedge$
$\quad$ A3.4.2 $\wedge$ A3.4.3 $\wedge\ I_x(w,m)\wedge$
$\quad \exists V'\, E'\, v_1\, v_2\, w[$A3.4.4 $\wedge$ A3.4.5$\wedge$
$\quad\quad \forall u\big(\neg V'(u) \vee ((u = v_1 \rightarrow u = 0) \wedge (u = v_2 \rightarrow u = 1) \wedge (u = w \rightarrow$ A3.4.6$)\wedge$
$\quad\quad\quad ((u \neq v_1 \wedge u \neq v_2) \rightarrow \exists y'(E'(y', u) \wedge$ A3.4.7$)))\big)\wedge$
$\quad\quad V_x(x') \rightarrow [$(A3.4.8) $\vee$ (A3.4.9 $\wedge \exists t(W_x(x', t)\ \wedge$ A3.4.10$))]\wedge$
$\quad\quad$ A3.4.11 $\wedge$ A3.4.12$]\big)$ where

- A3.4.1 expresses "$I_x$ is a bijection from $V_x$ to $\{1, \ldots, m\}$".
- A3.4.2 expresses "$v$ and $w$ are the first and last nodes of $(V_x, E_x)$, respectively".
- A3.4.3 expresses "$I_x(v, 1)$".
- A3.4.4 expresses "$(V', E')$ is a linear graph".
- A3.4.5 expresses "$v_1, v_2$ and $w$ are the 1-st, 2-nd and last nodes in $(V', E')$, respectively".
- A3.4.6 expresses "$\exp(2, m-1, u)$".
- A3.4.7 expresses "$\text{times}(2, y', u)$".
- A3.4.8 expresses "$B_x(x') = 0$" $\wedge$ "$W_x(x') = 0$".
- A3.4.9 expresses "$B_x(x') = 1$" $\wedge$ "$\text{sum}(n_x, I_x(x'), m)$".
- A3.4.10 expresses "$\exp(2, n_x, t)$"
- A3.4.11 expresses "$F(x) = W_x(x_1) + W_x(x_2) + \cdots + W_x(x_m)$ for $x_i$ the $i$-th node in $(V_x, E_x)$".

- A3.4.12 expresses "$W_x$ is a function from $V_x$ to $V'$".

Finally, we note that A3.4.11 can be expressed as follows.

$$\exists U_x\big(\text{A3.4.11.1} \wedge \forall x'\big(\neg V_x(x') \vee (\text{A3.4.11.2} \wedge \text{A3.4.11.3} \wedge (\text{A3.4.11.4} \rightarrow \exists x''(E_x(x'', x') \wedge \text{A3.4.11.5})))\big)\big) \text{ where}$$

- A3.4.11.1 expresses "$U_x$ is a function from $V_x$ to $V$".
- A3.4.11.2 expresses "if $x'$ is the first node in $(V_x, E_x)$ then $U_x(x') = W_x(x')$".
- A3.4.11.3 expresses "if $x'$ is the last node in $(V_x, E_x)$ then $U_x(x') = F(x)$".
- A3.4.11.4 expresses "$x'$ is not the first node in $(V_x, E_x)$".
- A3.4.11.5 expresses "$\text{sum}(U_x(x''), W_x(x'), U_x(x'))$".

## *4.2 Second Strategy*

The second strategy to define the class of hypercube graphs can be described in two steps.

i. To identify every node $x$ in the input graph $\mathbf{G}$ with a different subset $S_x$ of a set $V' \subset V$ of cardinality $\log_2 |V|$, making sure that every subset of $V'$ is assigned to some node of $\mathbf{G}$.

ii. To check that for every pair of nodes $x$ and $y$ in $\mathbf{G}$, there is an edge between $x$ and $y$ iff $S_x$ can be obtained from $S_y$ by adding or removing a single element.

In second-order logic we can express this strategy as follows.

$$\varphi_2 \equiv \exists R\big(\exists V'\big(\text{A1} \wedge \forall S(\text{A2} \rightarrow (\exists x(\text{A3} \wedge \text{A4}))) \wedge \exists z(\text{A5})\big) \wedge \forall xy((E(x, y) \wedge E(y, x)) \leftrightarrow \text{A6})\big)$$

where

- A1 expresses "$V' \subset V \wedge V' \neq \emptyset$".
- A2 expresses "$S \subseteq V' \wedge S \neq \emptyset$".
- A3 expresses "$x$ is identified with $S$ via $R$".
  $\text{A3} \equiv \forall v(R(x, v) \leftrightarrow S(v))$
- A4 expresses "no other node $y \neq x$ can be identified with $S$ via $R$".
  $\text{A4} \equiv \neg\exists y(x \neq y \wedge \forall v(R(y, v) \leftrightarrow S(v)))$
- A5 expresses "all nodes, with the only exception of node $z$, are identified with some nonempty subset of $V'$ via R".
  $\text{A5} \equiv \neg\exists v\big(R(z, v)\big) \wedge \forall z'\big(z \neq z' \rightarrow \exists S(\text{A5.1} \wedge \forall v(R(z', v) \leftrightarrow S(v)))\big)$ where
  – A5.1 expresses "$S \neq \emptyset \wedge S \subseteq V'$".
- A6 expresses "the set $S_x$ identified with $x$ can be obtained from the set $S_y$ identified with $y$ by adding or removing a single element".
  $\text{A6} \equiv \exists v\big(\big((R(x, v) \wedge \neg R(y, v)) \vee (R(y, v) \wedge \neg R(x, v))\big) \wedge \forall v'\big(v' \neq v \rightarrow (R(x, v') \leftrightarrow R(y, v'))\big)\big)$

REMARK 4.1
The formula $\varphi_2$ that expresses the second strategy has a prefix of second-order quantifiers of the form $\exists R \exists V' \forall S$. Thus, it is in the class $\Sigma_2^1$. The existence of a formula in $\Sigma_1^1$ that expresses this second strategy is unlikely, since we must express that *every subset* $S$ is identified with some node in the graph. On the other hand, the formula $\varphi_1$ that expresses the first strategy, while considerably more cumbersome than $\varphi_2$, only uses existential second-order quantifiers and can be translated in a rather straightforward way into an equivalent $\Sigma_1^1$ formula. That is, we could transform the current quantification schema of the form

$$\forall xy\big(\exists V_x^1 E_x^2 B_x^2 V_y^1 E_y^2 B_y^2 \dots \exists W_x^2 U_x^2 W_y^2 U_y^2 \dots\big),$$

where the superindices added to the relation variables denote their arity, into an schema of the form

$$\big(\exists V_x^2 E_x^3 B_x^3 V_y^2 E_y^3 B_y^3 \dots \exists W_x^3 U_x^3 W_y^3 U_y^3 \dots\big),$$

where the prefix "$\forall xy$" is eliminated and the arity of every relation variable is increased in 1, so that we can incorporate all nodes. Thus, for instance, every set $V_x^1$ corresponding to some node $x$ in a graph $\mathbf{G}$ is now encoded in the binary relation $V_x^2$ in such a way that $V_x^1 = \{y|(x,y) \in V_x^2\}$. Then, we can simply express that the set $\{x|(x,y) \in V_x^2\}$ contains every node in the graph $\mathbf{G}$. Moreover, we can now omit $V_y, E_y, B_y, W_y$ and $U_y$, since for every pair of nodes $x$ and $y$, their corresponding sets $V_x^1$ and $V_y^1$ will be both encoded into the binary relation $V_x^2$, and something similar will happen for the relations $E$, $B$, $W$ and $U$.

This is an important consideration since by Fagin-Stockmeyer characterization of the polynomial-time hierarchy [13] $\Sigma_1^1$ captures NP while $\Sigma_2^1$ captures $\mathrm{NP}^{\mathrm{NP}}$.

## 5 Quantified Boolean Formulae

A *Boolean variable* is any symbol to which we can associate the truth values 0 and 1. Let $V$ be a countable set of Boolean variables. The class of *Boolean formulae* over $V$ is the smallest class which is defined by:

- The Boolean constants 0 and 1 are Boolean formulae.
- Every Boolean variable $x$ in $V$ is a Boolean formula.
- If $\varphi$ and $\psi$ are Boolean formulae then $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$ and $\neg(\varphi)$ are Boolean formulae.

The semantics of the Boolean formulae is given by the well-known semantics of the propositional logic.

A *quantified Boolean formula* over $V$, as defined by the influential Garey and Johnson book on the theory of NP-Completeness [6], is a formula of the form

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n(\varphi),$$

where $\varphi$ is a Boolean formula over $V$, $n \geq 0$, $x_1, \dots, x_n \in V$ and, for $1 \leq i \leq n$, $Q_i$ is either "$\exists$" or "$\forall$". A variable that occurs in the Boolean formula but does not occur in the prefix of quantifiers is called a *free variable*. We call QBF the set

of quantified Boolean formulae without free variables. As usual, for $k \geq 1$, $\mathrm{QBF}_k$ denotes the fragment of QBF which consists of those formulae which start with an existential block and have $k$ alternating blocks of quantifiers. Let $X \subset V$ be a finite set of Boolean variables, we assume w.l.o.g. that a formula in $\mathrm{QBF}_k$ over $X$ is of the form

$$\exists \bar{x}_1 \forall \bar{x}_2 \ldots Q \bar{x}_k(\varphi),$$

where for $1 \leq i \leq k$, $\bar{x}_i = (x_{i1}, \ldots, x_{il_i})$ is a vector of $l_i$ different variables from $X$, $\exists \bar{x}_i$ denotes a block of $l_i$ quantifiers of the form $\exists x_{i1}, \ldots, \exists x_{il_i}$, $\forall \bar{x}_i$ denotes a block of $l_i$ quantifiers of the form $\forall x_{i1}, \ldots, \forall x_{il_i}$, $\varphi$ is a (quantifier free) Boolean formula over $X$, $Q$ is "$\exists$" if $k$ is odd and "$\forall$" if $k$ is even, and the sets $X_1, \ldots, X_k$ of variables in $\bar{x}_1, \ldots, \bar{x}_k$, respectively, form a partition of $X$.

We define next the notion of satisfiability of quantified Boolean formulae. But first we introduce the concept of alternating valuations which uses rooted binary trees to represent all possible valuations for a given formula, and paths from the root to the leaves of such trees to represent individual valuations. This unusual way of representing valuations is motivated by the way in which we express in second-order logic the satisfiability problem for the classes $\mathrm{QBF}_k$.

Let $\mathbf{T}_v$ be a rooted binary tree of vocabulary $\sigma_{\mathbf{T}_v} = \{E, B, 0, 1\}$. That is, $\mathbf{T}_v$ is a maximally connected acyclic digraph in which every vertex has at most two child vertices and, except for the root, has a unique parent. Here, 0 and 1 are constant symbols which are interpreted as truth values and $B^{\mathbf{T}_v}$ is a total function which assigns a truth value $0^{\mathbf{T}_v}$ or $1^{\mathbf{T}_v}$ to each vertex in $V$. We say that $\mathbf{T}_v$ is an *alternating valuation* if the following holds:

- Every leaf of $\mathbf{T}_v$ is at the same depth $d$.
- All vertices at a given depth, i.e., in the same level, have the same out-degree.
- If two vertices $a, b \in V$ are siblings, then $B^{\mathbf{T}_v}(a) \neq B^{\mathbf{T}_v}(b)$.

Let $\varphi \equiv \exists \bar{x}_1 \forall \bar{x}_2 \ldots Q \bar{x}_k(\psi)$ be a formula in $\mathrm{QBF}_k$, where $Q$ is "$\exists$" if $k$ is odd and "$\forall$" if $k$ is even, and let $l_j$ for $1 \leq j \leq k$ be the length of the $j$-th alternating block of quantifiers. We say that an alternating valuation $\mathrm{T}_v$ is *applicable* to $\varphi$, if the depth of $\mathrm{T}_v$ is $l_1 + \cdots + l_k - 1$ and for every $1 \leq i \leq l_1 + \cdots + l_k$, it holds that:

- All vertices at depth $i-1$ have no siblings if $1 \leq i \leq l_1$ or $l_1+l_2+1 \leq i \leq l_1+l_2+l_3$ or $\cdots$ or $l_1 + l_2 + \cdots + l_{k'-1} + 1 \leq i \leq l_1 + l_2 + \cdots + l_{k'}$, where $k' = k$ if the $k$-th block of quantifiers is existential and $k' = k - 1$ otherwise.
- All vertices at depth $i - 1$ have exactly one sibling if $l_1 + 1 \leq i \leq l_1 + l_2$ or $l_1 + l_2 + l_3 + 1 \leq i \leq l_1 + l_2 + l_3 + l_4$ or $\cdots$ or $l_1 + l_2 + \cdots + l_{k''-1} + 1 \leq i \leq l_1 + l_2 + \cdots + l_{k''}$, where $k'' = k$ if the $k$-th block of quantifiers is universal and $k'' = k - 1$ otherwise.

Let $\gamma = \exists \bar{x}_1 \forall \bar{x}_2 \ldots Q \bar{x}_k(\varphi)$ be a formula in $\mathrm{QBF}_k$ over $X$, and let $\mathbf{T}_v$ be an alternating valuation applicable to $\gamma$. A *leaf valuation* $\mathbf{L}_v$ is a linear subgraph of $\mathbf{T}_v$ of vocabulary $\sigma_{\mathbf{T}_v}$ which corresponds to a path from the root to a leaf in $\mathbf{T}_v$. Let $v$ be a mapping from the set of variables $X$ to $\{0, 1\}$, i.e., a Boolean assignment, such that for $x_i \in X$ the $i$-th variable in the prefix of quantifiers of $\gamma$, it holds that $v(x_i) = 1$ iff $B^{\mathbf{L}_v}(n_i) = 1^{\mathbf{L}_v}$ for $n_i$ the $i$-th node in the linear order induced by $E^{\mathbf{L}_v}$. We say that $\mathbf{L}_v$ *satisfies* $\gamma$, written $\mathbf{L}_v \models \gamma$, if the Boolean assignment $v$ satisfies $\varphi$. That is, if $\varphi$

is a Boolean variable $x_i$ in $X$, then $\mathbf{L}_v \models \varphi$ if $v(x_i) = 1$; if $\varphi = \neg(\psi)$, then $\mathbf{L}_v \models \varphi$ if $\mathbf{L}_v \not\models \psi$ (i.e., if it is not the case that $\mathbf{L}_v \models \psi$); if $\varphi = (\psi \vee \alpha)$, then $\mathbf{L}_v \models \varphi$ if either $\mathbf{L}_v \models \psi$ or $\mathbf{L}_v \models \alpha$; and if $\varphi = (\psi \wedge \alpha)$, then $\mathbf{L}_v \models \varphi$ if both $\mathbf{L}_v \models \psi$ and $\mathbf{L}_v \models \alpha$. Finally, we say that the alternating valuation $\mathbf{T}_v$ *satisfies* $\gamma$ if every leaf valuation $\mathbf{L}_v$ of $\mathbf{T}_v$ satisfies $\gamma$.

A Boolean formulae $\varphi$ in $\mathrm{QBF}_k$ is *satisfiable* if and only if there is an alternating valuation $\mathbf{T}_v$ which satisfies $\varphi$; otherwise $\varphi$ is *unsatisfiable*. $\mathrm{SATQBF}_k$ is the set of $\mathrm{QBF}_k$ formulae that are satisfiable. $\mathrm{SATQBF} = \bigcup_{k\geq 1} \mathrm{SATQBF}_k$.

It is well known that $\mathrm{SATQBF}_k$ is complete for the level $\Sigma_k^p$ of the polynomial-time hierarchy (see [6, 1] among others sources). It is also well known that second-order logic captures the polynomial-time hierarchy. In fact, there is an exact correspondence between the prenex fragments of second-order logic with up to $k$ alternations of quantifiers $\Sigma_k^1$ and the levels $\Sigma_k^p$ of the polynomial time hierarchy [13]. Thus, for every $k$, $\mathrm{SATQBF}_k$ can be defined in second-order logic, in fact, it can even be defined in $\Sigma_k^1$. Regarding SATQBF, we note that it is PSPACE-complete [13]. Since existential third-order logic captures $\mathrm{NTIME}(2^{n^{\mathcal{O}(1)}})$ (see [7]) and $\mathrm{PSPACE} \subseteq \mathrm{DTIME}(2^{n^{\mathcal{O}(1)}}) \subseteq \mathrm{NTIME}(2^{n^{\mathcal{O}(1)}})$, we know that SATQBF can be defined in existential third-order logic. In the following sections we present a second-order formula that defines $\mathrm{SATQBF_k}$ and a third-order formula that defines SATQBF, respectively.

## 6 $\mathrm{SATQBF_k}$ in Second-Order Logic

Following a top-down approach, we present a detailed construction of a second-order formula that defines $\mathrm{SATQBF}_k$. But first, we need to fix an encoding of quantified Boolean formulae as relational structures.

There is a well-known correspondence between words and finite structures. Let $A$ be a finite alphabet and let $\pi(A)$ be the vocabulary $\{\leq\} \cup \{R_a : a \in A\}$, where $\leq$ is a binary relation symbol and the $R_a$ are unary relation symbols. We can identify any word $v = a_1 \dots a_n$ in $A^*$ with a $\pi(A)$-structure $\mathbf{B}$, where the cardinality of $B$ equals the length of $v$, $\leq^{\mathbf{B}}$ is a total order on $\mathbf{B}$, and, for each $R_a \in \pi(A)$, $R_a^{\mathbf{B}}$ contains the positions in $v$ carrying an $a$,

$$R_a^{\mathbf{B}} = \{b \in B : \text{for some } j \ (1 \leq j \leq n), \\ b \text{ is the } j\text{-th element in the order } \leq^{\mathbf{B}} \text{ and } a_j = a\}$$

Such structures are usually known as *word models* for $v$ ([3]). As any two word models for $v$ are isomorphic, we can speak of *the* word model for $v$.

Note that we can represent Boolean variables of the form $x_n$ by using a symbol "$X$" followed by a sequence of $n$ symbols "|". For instance, we can write $X|||$ for $x_3$. Thus using *word models*, every quantified Boolean formula $\varphi$ can be viewed as a finite relational structure $G_\varphi$ of the following vocabulary.

$$\pi = \{\leq, P_\neg, P_\vee, P_\wedge, P_\exists, P_\forall, P_(, P_), P_X, P_|\}$$

Example 6.1
If $\varphi$ is the quantified Boolean formula $\exists x_1 \forall x_2((\neg x_1) \vee x_2)$, which using our notation for the variables corresponds to $\exists X|\forall X||((\neg X|) \vee X||)$, then the following $\pi$-structure $\mathbf{G}_\varphi$ (note that $\mathbf{G}_\varphi$ is a linear graph) where $G_\varphi = \{1, \dots, 18\}$, $\leq^{\mathbf{G}_\varphi}$ is a total order

on $\mathbf{G}_\varphi$, $P_\neg^{\mathbf{G}_\varphi} = \{10\}$, $P_\vee^{\mathbf{G}_\varphi} = \{14\}$, $P_\wedge^{\mathbf{G}_\varphi} = \emptyset$, $P_\exists^{\mathbf{G}_\varphi} = \{1\}$, $P_\forall^{\mathbf{G}_\varphi} = \{4\}$, $P_($^{\mathbf{G}_\varphi} = \{8, 9\}$, $P_)^{\mathbf{G}_\varphi} = \{13, 18\}$, $P_X^{\mathbf{G}_\varphi} = \{2, 5, 11, 15\}$, $P_|^{\mathbf{G}_\varphi} = \{3, 6, 7, 12, 16, 17\}$, encodes $\varphi$.

We show next how to build a second-order logic formula $\varphi_{\text{SATQBF}_k}$ such that, given a relational structure $\mathbf{G}_\varphi$ of vocabulary $\pi$, it holds that $\mathbf{G}_\varphi \models \varphi_{\text{SATQBF}_k}$ iff the quantified Boolean formula $\varphi$ represented by $\mathbf{G}_\varphi$, is satisfiable. That is, we show next how to build a second-order formula $\varphi_{\text{SATQBF}_k}$ of vocabulary $\pi$ that defines $\text{SATQBF}_k$. As mentioned earlier, we follow a top-down approach for the construction of this formula. At the highest level of abstraction, we can think of $\varphi_{\text{SATQBF}_k}$ as a second-order formula that expresses the following:

"There is an alternating valuation $\mathbf{T}_v$ applicable to $\varphi$ that satisfies $\varphi$". (A)

Recall that an alternating valuation $\mathbf{T}_v$ satisfies $\varphi$ iff every leaf valuation $\mathbf{L}_v$ of $\mathbf{T}_v$ satisfies the quantifier-free part $\varphi'$ of $\varphi$. Also recall that every leaf valuation $\mathbf{T}_v$ corresponds to a Boolean assignment $v$. Thus, if $\varphi = \exists \bar{x}_1 \forall \bar{x}_2 \ldots Q \bar{x}_k(\varphi')$, where for $1 \leq i \leq k$, $\bar{x}_i = (x_{i1}, \ldots, x_{il_i})$, $Q$ is "$\exists$" if $k$ is odd and "$\forall$" if $k$ is even, $X_1, \ldots, X_k$ are the set of variables in $\bar{x}_1, \ldots, \bar{x}_k$, respectively, and $\varphi'$ is a (quantifier free) Boolean formulae over $X = X_1 \cup \cdots \cup X_k$, then the expression in (A) can be divided in two parts:

AVS1 (Alternating Valuation that Satisfies $\varphi$, Part 1) which expresses

"There is a partial Boolean assignment $v_1$ on $X_1$,
such that for all partial Boolean assignments $v_2$ on $X_2$,
...,
there is (or "for all" if k is even) a partial Boolean assignment $v_k$ on $X_k$".

AVS2 (Alternating Valuation that Satisfies $\varphi$, Part 2) which expresses

"The Boolean assignment $v = v_1 \cup v_2 \cup \cdots \cup v_k$ satisfies the (quantifier free) Boolean formula $\varphi'$".

For each partial Boolean assignment $v_i$ $(1 \leq i \leq k)$, we use a second-order variable $V_i$ of arity one and two second-order variables $E_i$ and $B_i$ of arity two, to store the encoding of each $v_i$ as a linear graph $G_i = (V_i, E_i)$ with an associated function $B_i : V_i \to \{0, 1\}$ (see Figure 6.1). Correspondingly, we use a second-order variable $V_t$ of

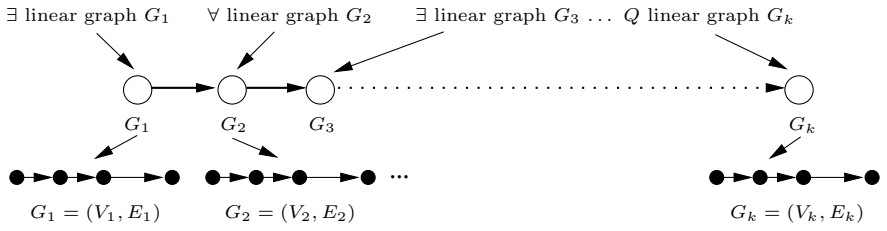


Figure 6.1

arity one and two second-order variables $E_t$ and $B_t$ of arity two, to store the encoding

of each Boolean assignment $v$ (leaf valuation $\mathbf{T}_v$) as a linear graph $G_t = (V_t, E_t)$ with an associated function $B_t : V_t \to \{0, 1\}$. Figure 6.2 illustrates an alternating valuation applicable to $\varphi$ and its corresponding encoding.

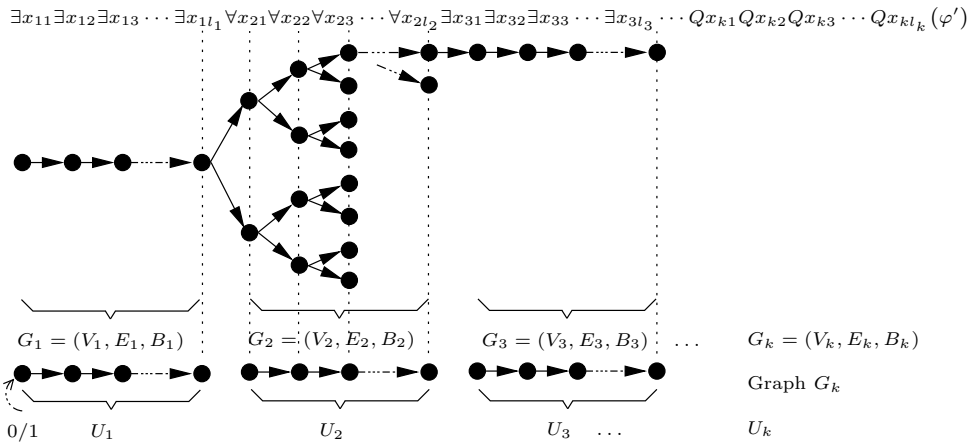


Figure 6.2

In the next subsection we describe the process followed to build a second-order formula to express Statement AVS1. Then we describe in Subsection 6.2, the corresponding process for Statement AVS2.

## *6.1 Expressing Statement AVS1*

Let $k_\exists$ and $k_\forall$ be the index of the last existential quantifier block and the last universal quantifier block, respectively, in the prefix of $k$ blocks of quantifiers of $\varphi$. We can express Statement AVS1 as follows:

$$\exists V_1 E_1 B_1 \forall V_2 E_2 B_2 \cdots Q_k V_k E_k B_k \exists V_t E_t B_t U_1, U_2, \ldots, U_k \Big( \text{A1} \wedge \text{A2} \wedge \text{A3} \wedge \text{A4} \wedge \text{A5} \wedge \big( (\text{A6} \wedge \text{A7} \wedge \text{A8} \wedge \text{A9} \wedge \text{A10} \wedge \text{A11}) \rightarrow \text{AVS2} \big) \Big) \text{ where}$$

- A1 expresses "$G_t = (V_t, E_t)$ is a linear graph".
- A2 expresses "The length of $G_t$ equals the number of variables that appear in the prefix of quantifiers of $\varphi$".
- A3 expresses "$G_1 = (V_1, E_1), G_3 = (V_2, E_2), \ldots, G_{k_\exists} = (V_{k_\exists}, L_{k_\exists})$ are linear graphs".
- A4 expresses "$B_1 : V_1 \to \{0, 1\}$, $B_3 : V_3 \to \{0, 1\}$, ..., $B_{k_\exists} : V_{k_\exists} \to \{0, 1\}$ are total functions".
- A5 expresses "The lengths of the linear graphs $G_1, G_3, \ldots, G_{k_\exists}$ equal the lengths of their corresponding blocks of quantifiers in $\varphi$".
- A6 expresses "$V_1, V_2, \ldots, V_k$ are pairwise disjoint sets".
- A7 expresses "$G_2 = (V_2, E_2), G_4 = (V_4, E_4), \ldots, G_{k_\forall} = (V_{k_\forall}, L_{k_\forall})$ are linear graphs"

- A8 expresses "$B_2 : V_2 \to \{0,1\}$, $B_4 : V_4 \to \{0,1\}$, ..., $B_{k_\forall} : V_{k_\forall} \to \{0,1\}$ are total functions"
- A9 expresses "The lengths of the linear graphs $G_2, G_4, \ldots, G_{k_\forall}$ equal the lengths of their corresponding blocks of quantifiers in $\varphi$".
- A10 expresses "$U_1$ is a total injection from $G_1$ to the first part of $G_t$ and $U_2$ is a total injection from $G_2$ to the second part of $G_t$ ...and $U_k$ is a total injection from $G_k$ to the $k$-th part of $G_t$".
- A11 expresses "$B_t : V_t \to \{0,1\}$ is a total function that *coincides* with $B_1$, $B_2$, ..., $B_k$".
- AVS2 expresses Statement AVS2 as described in Subsection 6.2.

Next, we discuss how to write the sub-formulae A1–A11 in second-order logic.

A1. This is expressed by the auxiliary formula LINEAR$(V_t, E_t)$, which is defined in Subsection 6.3 below.

A2. This is implied by the following statement which is expressed in further detail in Subsection 6.2.1 (A).
"There is a partial surjective injection $V_p$ from the quantifier prefix of $\varphi$ to $G_t$, which maps every $X$ in the prefix to its corresponding node in $G_t$, and which preserves $\leq^{\mathbf{G}_\varphi}$ and $E_t$".

A3. LINEAR$(V_1, E_1) \wedge$ LINEAR$(V_3, E_3) \wedge \cdots \wedge$ LINEAR$(V_{k_\exists}, E_{k_\exists})$, where the subformulae LINEAR$(V_i, E_i)$ are as defined in Subsection 6.3.

A4. $\forall t, p, p' \Big( \bigwedge_{i=1,3,\ldots,k_\exists} (\text{A4.1} \wedge \text{A4.2} \wedge \text{A4.3}) \Big)$
- A4.1 expresses "$B_i$ is a function".
  A4.1 $\equiv ((B_i(t,p) \wedge B_i(t,p')) \to p = p')$
- A4.2 expresses "$B_i$ is total".
  A4.2 $\equiv (V_i(t) \to \exists p(B_i(t,p)))$
- A4.3 expresses "the range of $B_i$ is $\{0,1\}$".
  A4.3 $\equiv (B_i(t,p) \to (p = 1 \vee p = 0))$
  where $p = 0$ and $p = 1$ have the obvious meaning and are defined in Subsection 6.3.

A5. If $k_\exists \neq k$, then

$\bigwedge_{1,3,\ldots,k_\exists} (\exists L' v_1 v_2 \ldots v_{k_\exists} v_{k_\exists+1} (\alpha_{k_\exists} \wedge \zeta_i))$

where $\alpha_{k_\exists}$ is the formula template $\alpha_i$ instantiated with $i = k_\exists$.

If $k_\exists = k$, then

$\big( \bigwedge_{1,3,\ldots,k_\exists-2} (\exists L' v_1 v_2 \ldots v_{k_\exists-1} (\alpha_{k_\exists-2} \wedge \zeta_i)) \big) \wedge \exists L' v_1 v_2 \ldots v_k v_e (\beta_1 \wedge \beta_2 \wedge \beta_3)$

where $\alpha_{k_\exists-2}$ is the formula template $\alpha_i$ instantiated with $i = k_\exists - 2$ (Note that $k_\exists - 2$ is the previous to the last existential block, and the subformulae $\beta_1, \beta_2$ and $\beta_3$ take care of the last block of quantifiers).

Next, we define the subformulae $\alpha_i$, $\beta_1$ $\zeta_i$, $\beta_2$ and $\beta_3$ in the listed order. For their definitions we use an auxiliary formula PATH$_\leq(x, y)$ which is in turn defined in Subsection 6.3 below, and which expresses "the pair $(x, y)$ is in the transitive closure of the relation $\leq$".

The subformula $\alpha_i$ is satisfied if, for $1 \leq j \leq i$, $v_j$ is the position of the first quantifier of the $j$-th block (when $i$ is not the last block of quantifiers).

$$\begin{aligned}\alpha_i \equiv \big(&P_\exists(v_1) \wedge P_\forall(v_2) \wedge \ldots \wedge P_Q(v_{i+1}) \wedge \neg\exists x(x \neq v_1 \wedge x \leq v_1) \wedge \\ &\mathrm{PATH}_\leq(v_1, v_2) \wedge \mathrm{PATH}_\leq(v_2, v_3) \wedge \cdots \wedge \mathrm{PATH}_\leq(v_i, v_{i+1}) \wedge \\ &\neg\exists x(\mathrm{PATH}_\leq(v_1, x) \wedge \mathrm{PATH}_\leq(x, v_2) \wedge x \neq v_1 \wedge x \neq v_2 \wedge P_\forall(x)) \wedge \\ &\neg\exists x(\mathrm{PATH}_\leq(v_2, x) \wedge \mathrm{PATH}_\leq(x, v_3) \wedge x \neq v_2 \wedge x \neq v_3 \wedge P_\exists(x)) \wedge \\ &\ldots \wedge \\ &\neg\exists x(\mathrm{PATH}_\leq(v_i, x) \wedge \mathrm{PATH}_\leq(x, v_{i+1}) \wedge x \neq v_i \wedge x \neq v_{i+1} \wedge P_Q(x))\big)\end{aligned}$$

where $P_Q$ is $P_\forall$ if $i$ is odd or $P_\exists$ if $i$ is even.

The subformula $\beta_1$ is satisfied if, for $1 \leq j \leq i$, $v_j$ is the position of the first quantifier of the $j$-th block.

$$\begin{aligned}\beta_1 \equiv \big(&P_\exists(v_1) \wedge P_\forall(v_2) \wedge \ldots \wedge \mathrm{P1}(v_k) \wedge P_|(v_e) \wedge \neg\exists x(x \leq v_1) \wedge \\ &\mathrm{PATH}_\leq(v_1, v_2) \wedge \mathrm{PATH}_\leq(v_2, v_3) \wedge \cdots \wedge \mathrm{PATH}_\leq(v_k, v_e) \wedge \\ &\neg\exists x(\mathrm{PATH}_\leq(v_1, x) \wedge \mathrm{PATH}_\leq(x, v_2) \wedge x \neq v_1 \wedge x \neq v_2 \wedge P_\forall(x)) \wedge \\ &\neg\exists x(\mathrm{PATH}_\leq(v_2, x) \wedge \mathrm{PATH}_\leq(x, v_3) \wedge x \neq v_2 \wedge x \neq v_3 \wedge P_\exists(x)) \wedge \\ &\ldots \wedge \\ &\neg\exists x(\mathrm{PATH}_\leq(v_k, x) \wedge \mathrm{PATH}_\leq(x, v_e) \wedge x \neq v_k \wedge x \neq v_e \wedge \mathrm{P2}(x))\big)\end{aligned}$$

where P1 is $P_\exists$ if $k$ is odd or $P_\forall$ if $k$ is even, and P2 is $P_\forall$ if $k$ is odd or $P_\exists$ if $k$ is even.

When $i$ is not the index of the last block of quantifiers, the subformula $\zeta_i$ is satisfied if $L'$ is a bijection from the indices of the symbols $X$ in the $i$-th alternating block of quantifiers to $V_i$, which preserves $E_i$ and $\mathrm{Next}_X = \{(a,b) \in \leq^{\mathbf{G}_\varphi} \mid a$ and $b$ are indices of symbols in the $i$-th block$\wedge P_X(a) \wedge P_X(b) \wedge \forall c((a \leq c \wedge c \leq b) \rightarrow \neg P_X(c))\}$ (i.e., the order of appearance of the $X$'s in the $i$-th block of quantifiers in the prefix of $\varphi$). This is illustrated in Figure 6.3. Recall that we encode in $G_i = (V_i, E_i, B_i)$ a partial truth assignment for the variables in the $i$-th alternating block of quantifiers.

$\zeta_i \equiv \big(\mathrm{A5.1} \wedge \mathrm{A5.2} \wedge \mathrm{A5.3} \wedge \mathrm{A5.4} \wedge \mathrm{A5.5}\big)$ where

- A5.1 defines the "domain of $L'$".
  $\mathrm{A5.1} \equiv \forall x\big((\mathrm{PATH}_\leq(v_i, x) \wedge \mathrm{PATH}_\leq(x, v_{i+1}) \wedge x \neq v_{i+1} \wedge P_X(x)) \leftrightarrow \exists y(L'(x,y))\big)$
- A5.2 expresses "$L'$ is surjective".
  $\mathrm{A5.2} \equiv \forall y\big(V_i(y) \rightarrow \exists z(L'(z,y))\big)$
- A5.3 expresses "$L'$ preserves $\mathrm{Next}_X$ and $E_i$" which implies injectivity.
  $$\begin{aligned}\mathrm{A5.3} \equiv \forall s t s' t' \Big(&\big(L'(s,t) \wedge L'(s',t') \wedge s \neq s' \wedge \mathrm{PATH}_\leq(v_i, s) \wedge \mathrm{PATH}_\leq(s', v_{i+1}) \\ &\wedge \mathrm{PATH}_\leq(s, s') \wedge \neg\exists z(\mathrm{PATH}_\leq(s, z) \wedge \mathrm{PATH}_\leq(z, s') \wedge \\ &z \neq s \wedge z \neq s' \wedge P_X(z))\big) \rightarrow E_i(t, t')\Big)\end{aligned}$$
- A5.4 defines the "range of $L'$".
  $\mathrm{A5.4} \equiv \forall x y(L'(x,y) \rightarrow V_i(y))$

- A5.5 expresses "$L'$ is a function".
  $\text{A5.5} \equiv \forall xyz\big((L'(x,y) \wedge L'(x,z)) \rightarrow y = z\big)$

The subformula $\beta_2$ is satisfied if $L'$ is a bijection from the indices of the symbols $X$ in the $k$-th alternating block of quantifiers to $V_k$, which preserves $E_k$ and $\text{Next}_X$ (i.e., the order of appearance of the $X$'s in the $k$-th block of quantifiers in the prefix of $\varphi$).

$\beta_2 \equiv \big(\text{A5.1}' \wedge \text{A5.2}' \wedge \text{A5.3}' \wedge \text{A5.4}' \wedge \text{A5.5}'\big)$ where

- A5.1′ defines the "domain of $L'$".
  $\text{A5.1}' \equiv \forall x\big((\text{PATH}_{\leq}(v_k,x) \wedge \text{PATH}_{\leq}(x,v_e) \wedge P_X(x)) \leftrightarrow \exists y(L'(x,y))\big)$
- A5.2′ expresses "$L'$ is surjective".
  $\text{A5.2}' \equiv \forall y\big(V_k(y) \rightarrow \exists z(L'(z,y))\big)$
- A5.3′ expresses "$L'$ preserves $\text{Next}_X$ and $E_k$" which implies injectivity.
  $$\text{A5.3}' \equiv \forall st s't'\Big(\big(L'(s,t) \wedge L'(s',t') \wedge s \neq s' \wedge \text{PATH}_{\leq}(v_k,s) \wedge \text{PATH}_{\leq}(s',v_e) \wedge \text{PATH}_{\leq}(s,s') \wedge \neg\exists z(\text{PATH}_{\leq}(s,z) \wedge \text{PATH}_{\leq}(z,s') \wedge z \neq s \wedge z \neq s' \wedge P_X(z))\big) \rightarrow E_k(t,t')\Big)$$
- A5.4′ defines the "range of $L'$".
  $\text{A5.4}' \equiv \forall xy(L'(x,y) \rightarrow V_k(y))$
- A5.5′ expresses "$L'$ is a function".
  $\text{A5.5}' \equiv \forall xyz\big((L'(x,y) \wedge L'(x,z)) \rightarrow y = z\big)$

The last subformula $\beta_3$ is satisfied if $v_e$ is the last symbol "|" in the prefix of quantifiers of $\varphi$. We use $\text{SUC}_{\leq}(x,y)$ to denote that $x$ is the immediate successor of $y$ in the total order $\leq^{\mathbf{G}_\varphi}$. The formula that expresses $\text{SUC}_{\leq}(x,y)$ is defined in Subsection 6.3.

$$\beta_3 \equiv \Big(\forall v'\big(\text{SUC}_{\leq}(v_e,v') \rightarrow \neg P_{|}(v')\big) \wedge P_{|}(v_e) \wedge \forall v'\big(\text{PATH}_{\leq}(v_e,v') \rightarrow (\neg P_{\exists}(v') \wedge \neg P_{\forall}(v'))\big) \wedge \exists xyw\forall v'\big(P_X(x) \wedge P_Q(w) \wedge \text{SUC}_{\leq}(x,y) \wedge \text{SUC}_{\leq}(w,x) \wedge \text{PATH}_{\leq}(y,v_e) \wedge ((\text{PATH}_{\leq}(v',v_e) \wedge \text{PATH}_{\leq}(y,v')) \rightarrow P_{|}(v'))\big)\Big)$$

where $P_Q$ is $P_{\exists}$ if $k$ is odd, or $P_{\forall}$ if $k$ is even.

A6. Let $V_i \cap V_j = \emptyset$ denote $\forall x\big((V_i(x) \rightarrow \neg V_j(x)) \wedge (V_j(x) \rightarrow \neg V_i(x))\big)$, we can express that $V_1, V_2, \ldots, V_k$ are pairwise disjoint sets as follows.

$$(V_1 \cap V_2 = \emptyset) \wedge (V_1 \cap V_3 = \emptyset) \wedge (V_1 \cap V_4 = \emptyset) \wedge \cdots \wedge (V_1 \cap V_k = \emptyset) \wedge (V_2 \cap V_3 = \emptyset) \wedge (V_2 \cap V_4 = \emptyset) \wedge \cdots \wedge (V_2 \cap V_k = \emptyset) \wedge \ldots \wedge (V_{k-1} \cap V_k = \emptyset)$$

A7. $\text{LINEAR}(V_2,E_2) \wedge \text{LINEAR}(V_4,E_4) \wedge \cdots \wedge \text{LINEAR}(V_{k_\forall},E_{k_\forall})$,
where $\text{LINEAR}(V_i,E_i)$ is as defined in Subsection 6.3.

A8. $\forall t,p,p'\Big(\bigwedge_{i=2,4,\ldots,k_\forall}\big(\text{A8.1} \wedge \text{A8.2} \wedge \text{A8.3}\big)\Big)$ where

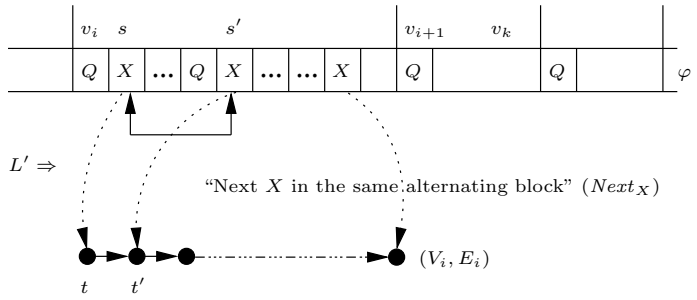


Figure 6.3

- A8.1 expresses "$B_i$ is a function".
  A8.1 $\equiv ((B_i(t,p) \wedge B_i(t,p')) \rightarrow p = p')$
- A8.2 expresses "$B_i$ is total".
  A8.2 $\equiv (V_i(t) \rightarrow \exists p(B_i(t,p)))$
- A8.3 expresses "the range of $B_i$ is $\{0,1\}$".
  A8.3 $\equiv (B_i(t,p) \rightarrow (p = 1 \vee p = 0))$
  where $p = 0$ and $p = 1$ have the obvious meaning and are defined in Subsection 6.3.

A9. If $k_\forall \neq k$, then

$\bigwedge_{2,4,\ldots,k_\forall} \left(\exists L' v_1 v_2 \ldots v_{k_\forall} v_{k_\forall+1} (\alpha_{k_\forall} \wedge \zeta_i)\right)$

where $\alpha_{k_\forall}$ is the formula template $\alpha_i$ instantiated with $i = k_\forall$.

If $k_\forall = k$, then

$\left(\bigwedge_{2,4,\ldots,k_\forall-2} \left(\exists L' v_1 v_2 \ldots v_{k_\forall-1} (\alpha_{k_\forall-2} \wedge \zeta_i)\right)\right) \wedge \exists L' v_1 v_2 \ldots v_k v_e (\beta_1 \wedge \beta_2 \wedge \beta_3)$

where $\alpha_{k_\forall-2}$ is the formula template $\alpha_i$ instantiated with $i = k_\forall - 2$ (Note that $k_\forall - 2$ is the previous to the last universal block, and the subformulae $\beta_1, \beta_2$ and $\beta_3$ take care of the last block of quantifiers).

The subformulae $\alpha_i$, $\zeta_i$, $\beta_1$, $\beta_2$ and $\beta_3$ are the same as in (A5).

A10. $\left(\text{A10.1} \wedge \bigwedge_{2 \leq i \leq k-1} \left(\text{A10.2.i}\right) \wedge \text{A10.3}\right)$ where

- A10.1 expresses "$U_1$ is a total injection from $V_1$ to $V_t$ such that: (a) preserves $E_1$ and $E_t$ and (b) $U_1$("first node in the order $E_1$") ="first node in the order $E_t$"".
- A10.2.i expresses "$U_i$ is a total injection from $V_i$ to $V_t$ such that: (a) preserves $E_i$ and $E_t$ and (b) $U_i$("first node in the order $E_i$") = $\text{SUC}_{E_t}(U_{i-1}$("last node in the order $E_{i-1}$"))".
- A10.3 expresses "$U_k$ is a total injection from $V_k$ to $V_t$ such that: (a) preserves $E_k$ and $E_t$ and (b) $U_k$("first node in order $E_k$") = $\text{SUC}_{E_t}(U_{k-1}$("last node in order $E_{k-1}$"))".

We describe next the second-order formula for A10.3 which is in turn illustrated in Figure 6.4. Note that the node labeled $x$ in Figure 6.4 corresponds to the last node in the linear graph $G_{k-1}$ and that $x$ is mapped by the function $U_{k-1}$ to the

node labelled $y$ in the linear graph $G_t$. Accordingly, $U_k$ maps the first node in the linear graph $G_k$ (i.e. the node labeled $u$), to the successor of node $y$ in $G_t$ (i.e. to the node labelled $t$).

A10.3 $\equiv \forall xytu\big(\text{A10.3.1} \wedge \text{A10.3.2} \wedge \text{A10.3.3} \wedge \text{A10.3.4}\big)$ where

- A10.3.1 expresses "$U_k$ is a total injection from $V_k$ to $V_t$".
  A10.3.1 $\equiv ((U_k(x,y) \wedge U_k(x,t)) \rightarrow y = t) \wedge$
  $((U_k(x,y) \wedge U_k(u,y)) \rightarrow x = u) \wedge$
  $(V_k(x) \rightarrow \exists y(U_k(x,y))) \wedge$
  $(U_k(x,y) \rightarrow (V_k(x) \wedge V_t(y)))$
- A10.3.2 expresses "preserves $E_t$".
  A10.3.2 $\equiv ((U_k(x,y) \wedge U_k(u,t) \wedge E_t(y,t)) \rightarrow E_k(x,u))$
- A10.3.3 expresses "preserves $E_k$".
  A10.3.3 $\equiv ((U_k(x,y) \wedge U_k(u,t) \wedge E_k(x,u)) \rightarrow E_t(y,t))$
- A10.3.4 expresses "$U_k$("first node in order $E_k$") = $\text{SUC}_{E_t}(U_{k-1}$("last node in order $E_{k-1}$"))".
  A10.3.4 $\equiv \big(\big(U_{k-1}(x,y) \wedge \neg\exists v(E_{k-1}(x,v)) \wedge E_t(y,t) \wedge \neg\exists v(E_k(v,u) \wedge V_k(u))\big)$
  $\rightarrow U_k(u,t)\big)$

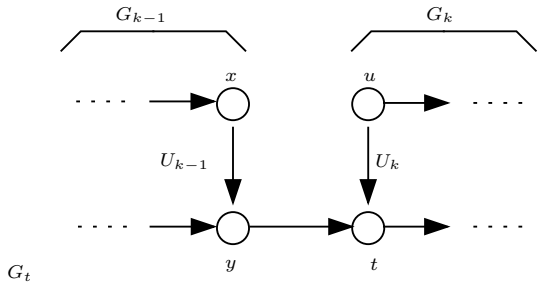


Figure 6.4

A11. $\forall xytpp'\big((B_1(t,p) \wedge U_1(t,y) \wedge B_t(y,p')) \rightarrow p = p'\big) \wedge$
$\forall xytpp'\big((B_2(t,p) \wedge U_2(t,y) \wedge B_t(y,p')) \rightarrow p = p'\big) \wedge$
$\ldots \wedge$
$\forall xytpp'\big((B_k(t,p) \wedge U_k(t,y) \wedge B_t(y,p')) \rightarrow p = p'\big)$

## *6.2 Expressing Statement AVS2*

Statement AVS2 can be rephrased as follows:

$\exists\, V_p\, C\, E_C\, ST\, E_{ST}\, M\, C_\wedge\, C_\vee\, C_\neg\, C_(\, C_)\, C_1\, C_0\, H_\phi\, \big(\text{AVS2.1} \wedge \text{AVS2.2}\big)$ where

- AVS2.1 expresses "There is a Boolean expression $\phi$ which is obtained from the quantifier-free part of $\varphi$ by replacing each occurrence of a variable by the corresponding truth value in $\{0,1\}$ assigned by the leaf valuation represented by $(G_t, B_t)$".

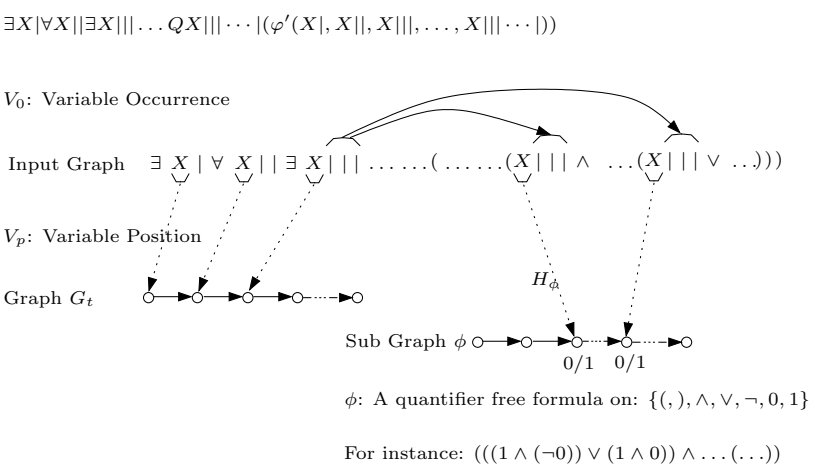


Figure 6.5

- AVS2.2 expresses "The Boolean expression $\phi$ evaluates to true".

We describe next how to express AVS2.1 and AVS2.2 in second-order logic.

### 6.2.1 Expressing AVS2.1

The idea is to define mappings to represent the relationships among the input graph $\mathbf{G}_\varphi$, the graph $G_t$ and the quantifier-free part of the input formulae. This is illustrated in Figure 6.5. We can express AVS2.1 as follows:

AVS2.1 $\equiv A \wedge B \wedge C$ where

- A expresses "$V_p$ is a partial bijection from the prefix of quantifiers of $\varphi$ (restricted to the $X$'s that appear in the prefix) to $V_t$, which maps every $X$ to its corresponding node in $G_t$, and which preserves $\leq^{\mathbf{G}_\varphi}$ and $E_t$".
  A $\equiv \forall xyz\big(\text{A1}\wedge\text{A2}\wedge\text{A3}\big)\wedge\forall sts't'\big(\text{A4}\big)$ where
  – A1 expresses "$V_p$ is a function".
    A1 $\equiv ((V_p(x,y)\wedge V_p(x,z))\rightarrow y=z)$
  – A2 expresses "$V_p$ is injective"
    A2 $\equiv ((V_p(x,y)\wedge V_p(z,y))\rightarrow x=z)$
  – A3 defines the "domain and range of $V_p$"
    A3 $\equiv \big((P_X(x)\wedge \text{PRED}_\leq(x,z)\wedge(P_\exists(z)\vee P_\forall(z)))\leftrightarrow\exists y(V_t(y)\wedge V_p(x,y))\big)$ where $\text{PRED}_\leq(x,z)$ denotes the subformula that expresses that $z$ is the strict predecessor of $x$ in the order $\leq^{\mathbf{G}_\varphi}$ (see Subsection 6.3).
  – A4 expresses "$V_p$ preserves $\leq^{\mathbf{G}_\varphi}$ and $E_t$".
    $(V_p(s,s')\wedge V_p(t,t')\wedge E_t(s',t'))\rightarrow$
    $\big(\text{PATH}_\leq(s,t)\wedge\forall z'\big((z'\neq s\wedge z'\neq t\wedge\text{PATH}_\leq(s,z')\wedge\text{PATH}_\leq(z',t))\rightarrow$
    $\neg P_X(z')\big)\big)$
- B expresses "$H_\phi$ is a partial surjective injection from the quantifier free part of $\varphi$ to the formula $\phi$, encoded as the first formula in $(C,E_C)$ (see Figures 6.7 and 6.8),

which maps every $X$ in the quantifier-free part of $\varphi$ to the corresponding position in the first formula in $(C, E_C)$ (i.e. $\phi$), which preserves $\wedge$, $\vee$, $\neg$, $(,)$, $\leq^{\mathbf{G}_\varphi}$ and $E_C$, and which ignores $|$".

$\mathrm{B} \equiv \forall x y_1 y_2 z_1 z_2 \big(\mathrm{B1} \wedge \mathrm{B2} \wedge \mathrm{B3} \wedge \mathrm{B4}\big) \wedge \forall x x' z y_1 y_2 z_1 z_2 \big(\mathrm{B5}\big)$ where

– B1 expresses "$H_\phi$ is a function".
$\mathrm{B1} \equiv \big((H_\phi(x, y_1, y_2) \wedge H_\phi(x, z_1, z_2)) \rightarrow$
$(y_1 = z_1 \wedge y_2 = z_2 \wedge \exists x'(P_($(x') \wedge \mathrm{PATH}_\leq(x', x)) \wedge C(y_1, y_2)))\big)$

– B2 expresses "$H_\phi$ is injective".
$\mathrm{B2} \equiv \big(H_\phi(x, y_1, y_2) \wedge H_\phi(z, y_1, y_2) \rightarrow x = z\big)$

– B3 expresses "the range of $H_\phi$ is the first formula in $(C, E_C)$".
$\mathrm{B3} \equiv \forall y_1' y_2' z_1' z_2' t_1' t_2' v' v_2 \big(\big(ST(v') \wedge \neg\exists y(E_{ST}(y, v')) \wedge$
$E_{ST}(v', v_2) \wedge M(v', y_1', y_2') \wedge M(v_2, z_1', z_2') \wedge$
$E_C(t_1', t_2', z_1', z_2') \wedge \mathrm{PATH}_{E_C}(y_1', y_2', y_1, y_2) \wedge$
$\mathrm{PATH}_{E_C}(y_1, y_2, t_1', t_2')\big) \rightarrow \exists x'(H_\phi(x', y_1, y_2))\big) \wedge$

– B4 expresses "the domain of $H_\phi$ corresponds to the quantifier free part of $\varphi$".
$\mathrm{B4} \equiv \big(\exists x'(P_($(x') \wedge \mathrm{PATH}_\leq(x', x)) \rightarrow \exists y_1' y_2'(H_\phi(x, y_1', y_2'))\big)$

– B5 expresses "$H_\phi$ preserves $\leq^{\mathbf{G}_\varphi}$ (ignoring "|"), $E_C$, $\wedge$, $\vee$, $(,)$ and $\neg$, and maps X to 0/1".
$\mathrm{B5} \equiv \big((H_\phi(x, y_1, y_2) \wedge H_\phi(z, z_1, z_2) \wedge E_C(y_1, y_2, z_1, z_2))$
$\rightarrow (\mathrm{SUC}_\leq(x, z) \vee (\mathrm{PATH}_\leq(x, z) \wedge \forall x'(\mathrm{PATH}_\leq(x, x') \wedge$
$\mathrm{PATH}_\leq(x', z) \wedge x' \neq x \wedge x' \neq z) \rightarrow P_|(x'))))\big) \wedge$
$\big(H_\phi(x, y_1, y_2) \rightarrow ((P_($(x) \wedge C_($(y_1, y_2)) \vee$
$(P_)(x) \wedge C_)(y_1, y_2)) \vee$
$(P_\wedge(x) \wedge C_\wedge(y_1, y_2)) \vee$
$(P_\vee(x) \wedge C_\vee(y_1, y_2)) \vee$
$(P_\neg(x) \wedge C_\neg(y_1, y_2)) \vee$
$(P_X(x) \wedge (C_0(y_1, y_2) \vee C_1(y_1, y_2)))))\big)$

- C expresses "for every bijection $V_0$ from "$|\cdots|$" in "$QX|\cdots|$" (where Q is "$\exists$" or "$\forall$") to "$|\cdots|$" in "$(\ldots X|\cdots|\ldots)$" that links a variable in the quantifier prefix of $\varphi$ with an occurrence of that variable in the quantifier-free part of it, the variable in the quantifier free part of $\varphi$ which corresponds to the function $V_0$ is replaced in $\phi$ by the value assigned to that variable by the leaf valuation $(G_t, B_t)$ (see Figures 6.5)". Note that in the formula below, $z_0$ represents the root in $\mathrm{dom}(V_0)$, $z_f$ represents the leaf in $\mathrm{dom}(V_0)$, $y_0$ represents the root in $\mathrm{ran}(V_0)$, and $y_f$ represents the leaf in $\mathrm{ran}(V_0)$ (see Figure 6.6). Also note that $\phi$ is encoded in $(C, E_C)$ starting in the node $M($"first node in $(ST, E_{ST})$"$)$ and ending in the node $E_C^{-1}(M($"second node in $(ST, E_{ST})$"$))$, and that it is equivalent to the quantifier-free part of $\varphi$ with the variables replaced by 0 or 1 according to the leaf valuation $(G_t, B_t)$ (this is further clarified in Subsection 6.2.2, also note Figures 6.7 and 6.8).

$\mathrm{C} \equiv \forall V_0 \exists z_0 y_0 z_f y_f z_0' y_0' z_f' y_f' \big((\mathrm{C1} \wedge \mathrm{C2} \wedge \mathrm{C3} \wedge \mathrm{C4} \wedge \mathrm{C5} \wedge \mathrm{C6}) \rightarrow \mathrm{C7}\big)$ where

– C1 expresses "$z_0$ is the root in $\mathrm{dom}(V_0)$, $z_f$ is the leaf in $\mathrm{dom}(V_0)$, $y_0$ is the root in $\mathrm{ran}(V_0)$ and $y_f$ is the leaf in $\mathrm{ran}(V_0)$".
$\mathrm{C1} \equiv V_0(z_0, y_0) \wedge \neg\exists z' y'(\mathrm{PRED}_\leq(z_0, z') \wedge V_0(z', y')) \wedge$
$V_0(z_f, y_f) \wedge \neg\exists z' y'(\mathrm{SUC}_\leq(z_f, z') \wedge V_0(z', y')) \wedge$
$\forall z'\big((\mathrm{PATH}_\leq(z_0, z') \wedge \mathrm{PATH}_\leq(z', z_f)) \rightarrow \exists y'(V_0(z', y'))\big) \wedge$
$\forall y'\big((\mathrm{PATH}_\leq(y_0, y') \wedge \mathrm{PATH}_\leq(y', y_f)) \rightarrow \exists z'(V_0(z', y'))\big)$

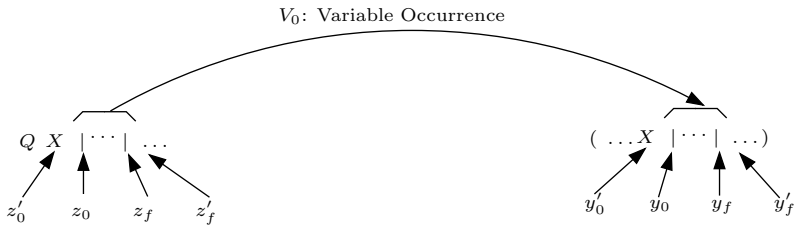


Figure 6.6

– C2 expresses "$V_0$ is a bijection from "$|\cdots|$" in "$QX|\cdots|$" to "$|\cdots|$" in "$(\ldots X|\cdots|\ldots)$" which preserves $\leq^{\mathbf{G}_\varphi}$".
  $\mathrm{C2} \equiv \forall xyvw\big((V_0(x,y) \to (P_|(x) \wedge P_|(y))) \wedge$
  $((V_0(x,y) \wedge V_0(x,v)) \to y = v) \wedge$
  $((V_0(x,y) \wedge V_0(w,y)) \to x = w) \wedge$
  $((V_0(x,y) \wedge V_0(v,w) \wedge \mathrm{SUC}_\leq(x,v)) \to \mathrm{SUC}_\leq(y,w))\big)$
– C3 expresses "$z'_0$ is the predecessor of the root in $\mathrm{dom}(V_0)$, i.e., it is the $X$ in the prefix of quantifiers".
  $\mathrm{C3} \equiv \mathrm{PRED}_\leq(z_0, z'_0) \wedge P_X(z'_0)$
– C4 expresses "$y'_0$ is the predecessor of the root in $\mathrm{ran}(V_0)$, i.e., it is the $X$ in the quantifier-free part".
  $\mathrm{C4} \equiv \mathrm{PRED}_\leq(y_0, y'_0) \wedge P_X(y'_0)$
– C5 expresses "$z'_f$ is the successor of the leaf in $\mathrm{dom}(V_0)$".
  $\mathrm{C5} \equiv \mathrm{SUC}_\leq(z_f, z'_f) \wedge \neg P_|(z'_f)$
– C6 expresses "$y'_f$ is the successor of the leaf in $\mathrm{ran}(V_0)$".
  $\mathrm{C6} \equiv \mathrm{SUC}_\leq(y_f, y'_f) \wedge \neg P_|(y'_f)$
– C7 expresses "$B_t(V_p(z'_0)) = H_\phi(y'_0)$".
  $\mathrm{C7} \equiv \forall\, x\, x' \,\big((V_p(z'_0, x) \wedge B_t(x, x')) \to$
  $\exists z_1 z_2 (H_\phi(y'_0, z_1, z_2) \wedge$
  $((\text{“}x' = 0\text{”} \wedge C_0(z_1, z_2)) \vee (\text{“}x' = 1\text{”} \wedge C_1(z_1, z_2)))))\big)$

### 6.2.2 Expressing AVS2.2

Now we need to check whether the formula $\phi$ built in the previous step, evaluates to true. The idea is to evaluate one connective at a time, and one pair of matching parenthesis at a time, until the final result becomes 1. Let us look at the example in Figure 6.7. Note that there are ten evaluation steps, which correspond to ten "operators" (i.e., either connectives or pairs of parenthesis). If there are at most $n$ symbols in $\phi$, that means that the whole evaluation process needs at most $n$ evaluation steps. This is the reason for using pairs of elements to represent the nodes of the graph $(C, E_C)$, and quadruples to represent the edges. This allows the whole evaluation process to take up to $n$ steps (where $n$ is the length of the input formula). In each step, we have a Boolean sentence on $\{0,1\}$ with up to $n$ symbols. Each node in the graph $(ST, E_{ST})$ represents one such formula, and the function $M$ (for Marker) is a pointer which tells us in which node in $(C, E_C)$ that formula begins. Note that in each evaluation step, either one or two symbols are removed from the formula at the

previous step. Figure 6.8 further illustrates the graphs (A) and (B) of Figure 6.7 with a horizontal orientation. Each evaluation step is called a stage. And the first symbol in each stage is given by the marker function $M$.

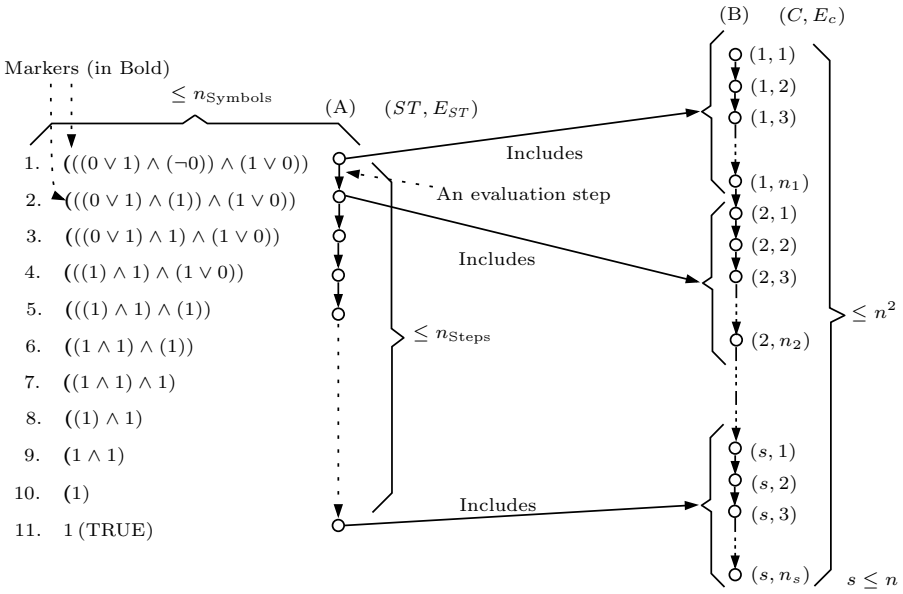


Figure 6.7

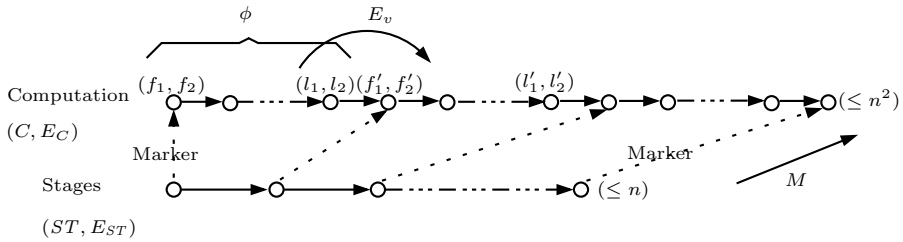


Figure 6.8

Based on this description, we can express AVS2.2 in Section 6.2 as follows:

A1 ∧ A2 ∧ A3 ∧ A4 ∧ A5 where

- A1 expresses "$(C, E_C)$ is a linear graph".
- A2 expresses "$(ST, E_{ST})$ is a linear graph".
- A3 expresses "$M : ST \to C$ is an injective and total function that preserves PATH in $E_{ST}$ and $E_C$".

- A4 expresses "$C_\wedge, C_\vee, C_\neg, C_(, C_), C_0, C_1$ are pairwise disjoint, and $C_\wedge \cup C_\vee \cup C_\neg \cup C_( \cup C_) \cup C_0 \cup C_1 = C$".
- A5 expresses "For every stage $x$, from stage $x$ to stage $x+1$, we need to follow the rules of evaluation (see Figure 6.7 part A). The formula in $(C, E_C)$ at stage $x+1$ is the same as the formula at stage $x$, except for one of three possible sorts of changes, which correspond to the cases (a), (b) and (c) of Figure 6.9".

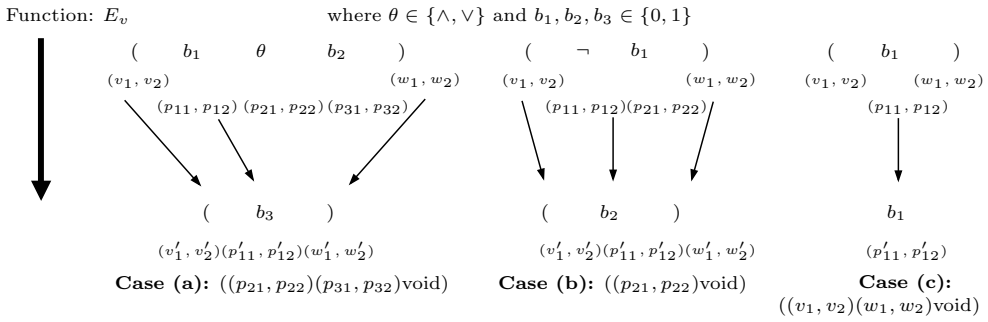


Figure 6.9

We describe next how to express A1–A5 above in second-order logic. See Section 6.3 for the auxiliary formulae used below.

A1 $\equiv$ LINEAR$(C, E_C)$

A2 $\equiv$ LINEAR$_2(ST, E_{ST})$

A3 $\equiv \forall s\, s'\, t_1\, t_2\, k_1\, k_2 \big(\text{A3.1} \wedge \text{A3.3} \wedge \text{A3.3} \wedge \text{A3.4}\big)$ where

- A3.1 expresses "$M$ is a function, $M : ST \to C$".
  A3.1 $\equiv ((M(s, t_1, t_2) \wedge M(s, k_1, k_2)) \to ((t_1 = k_1 \wedge t_2 = k_2) \wedge ST(s) \wedge C(t_1, t_2)))$
- A3.2 expresses "$M$ is injective".
  A3.2 $\equiv ((M(s, k_1, k_2) \wedge M(t_1, k_1, k_2)) \to s = t_1)$
- A3.3 expresses "$M$ is total".
  A3.3 $\equiv (ST(s) \to \exists t'_1 t'_2 (M(s, t'_1, t'_2)))$
- A3.4 expresses "$M$ preserves PATH in $E_{ST}$ and $E_C$".
  A3.4 $\equiv ((M(s, t_1, t_2) \wedge M(s', k_1, k_2) \wedge \text{PATH}_{ST}(s, s')) \to \text{PATH}_{E_C}(t_1, t_2, k_1, k_2))$

$$
\begin{aligned}
\text{A4} \equiv \forall s_1 s_2 \big( & (C_\wedge(s_1, s_2) \to \neg C_\vee(s_1, s_2)) \wedge (C_\wedge(s_1, s_2) \to \neg C_\neg(s_1, s_2)) \wedge \\
& (C_\wedge(s_1, s_2) \to \neg C_(s_1, s_2)) \wedge (C_\wedge(s_1, s_2) \to \neg C_)(s_1, s_2)) \wedge \\
& (C_\wedge(s_1, s_2) \to \neg C_0(s_1, s_2)) \wedge (C_\wedge(s_1, s_2) \to \neg C_1(s_1, s_2)) \wedge \cdots \big) \wedge \\
\forall s_1 s_2 \big( & C(s_1, s_2) \to (C_\wedge(s_1, s_2) \vee C_\vee(s_1, s_2) \vee C_\neg(s_1, s_2) \vee C_(s_1, s_2) \vee C_)(s_1, s_2) \vee \\
& C_0(s_1, s_2) \vee C_1(s_1, s_2))) \big) \wedge \\
\forall s_1 s_2 \big( & (C_\wedge(s_1, s_2) \to C(s_1, s_2)) \wedge (C_\vee(s_1, s_2) \to C(s_1, s_2)) \wedge \\
& (C_\neg(s_1, s_2) \to C(s_1, s_2)) \wedge (C_(s_1, s_2) \to C(s_1, s_2)) \wedge \\
& (C_)(s_1, s_2) \to C(s_1, s_2)) \wedge (C_0(s_1, s_2) \to C(s_1, s_2)) \wedge \\
& (C_1(s_1, s_2) \to C(s_1, s_2))\big)
\end{aligned}
$$

$\text{A5} \equiv \forall x\big(ST(x) \rightarrow \exists E_v\, f_1\, f_2\, l_1\, l_2\, f'_1\, f'_2\, l'_1\, l'_2\big(\alpha_d \vee \alpha_e \vee (\alpha_0 \wedge (\alpha_a \vee \alpha_b \vee \alpha_c))\big)\big)$ where

The function $E_v$ maps the formula at stage $x$ to the formula at stage $x+1$. The subformula $\alpha_d$ corresponds to the last transition, i.e., the transition to the last formula in $(C, E_C)$ ("0" or "1"). The subformula $\alpha_e$ corresponds to the last formula in $(C, E_C)$. The subformulae $\alpha_a$, $\alpha_b$ and $\alpha_c$ correspond to the three possible cases $(a)$, $(b)$ and $(c)$ as in Figure 6.9, according to which sort of operation is the one involved in the transition from the formula in stage $x$ to the next formula in $(C, E_C)$. Note that the transition to the last formula $\alpha_d$ is necessarily an instance of case $(c)$ in Figure 6.9. For case $(c)$ in Figure 6.9, $E_v$ is not total in its domain, since $(v_1, v_2)\big(\big()$ and $(w_1, w_2)\big()\big)$ are not mapped. For the last formula, $E_v$ is not injective, since $(f'_1, f'_2) = (l'_1, l'_2)$ (i.e., $f'_1 = l'_1$ and $f'_2 = l'_2$) (see Figure 6.11).

$\alpha_0 \equiv \text{A5.1} \wedge \text{A5.2} \wedge \text{A5.3} \wedge \text{A5.4}$ where

- A5.1 expresses "$x$ is not the leaf in $E_{ST}$, and it is not the predecessor of the leaf".
  $\text{A5.1} \equiv \exists y y_1(E_{ST}(x, y) \wedge E_{ST}(y, y_1))$
- A5.2 expresses "$E_v : C \rightarrow C$ is a partial injection mapping the formula in $(C, E_C)$ in stage $x$ to the formula in $(C, E_C)$ in stage $E_{ST}(x)$"
  $\text{A5.2} \equiv \forall s_1 s_2 t_1 t_2 k_1 k_2\big(((E_v(s_1, s_2, t_1, t_2) \wedge E_v(s_1, s_2, k_1, k_2)) \rightarrow ((t_1 = k_1 \wedge t_2 = k_2) \wedge C(s_1, s_2) \wedge C(t_1, t_2))) \wedge ((E_v(s_1, s_2, k_1, k_2) \wedge E_v(t_1, t_2, k_1, k_2)) \rightarrow (s_1 = t_1 \wedge s_2 = t_2))\big)$
- A5.3 expresses "$((f_1, f_2), (l_1, l_2))$ and $((f'_1, f'_2), (l'_1, l'_2))$ are the *delimiters* of the two formulae as in Figure 6.10".
  $\text{A5.3} \equiv M(x, f_1, f_2) \wedge \text{A5.3.1} \wedge \text{A5.3.2} \wedge 5.3.3$ where
  – A5.3.1 expresses "$M(E_{ST}(x), E_C(l_1, l_2))$".
  – A5.3.2 expresses "$E_C(l_1, l_2) = (f'_1, f'_2)$".
  – A5.3.3 expresses "$E_C^{-1}(M(E_{ST}(E_{ST}(x))), l'_1, l'_2)$".
- A5.4 expresses "$E_v$ maps nodes from the subgraph induced by $((f_1, f_2), (l_1, l_2))$ to the subgraph induced by $((f'_1, f'_2), (l'_1, l'_2))$".
  $\text{A5.4} \equiv \forall y_1 y_2 z_1 z_2\big(E_v(y_1, y_2, z_1, z_2) \rightarrow \big(\text{PATH}_{E_C}(f_1, f_2, y_1, y_2) \wedge \text{PATH}_{E_C}(y_1, y_2, l_1, l_2) \wedge \text{PATH}_{E_C}(f'_1, f'_2, z_1, z_2) \wedge \text{PATH}_{E_C}(z_1, z_2, l'_1, l'_2)\big)\big) \wedge E_v(f_1, f_2, f'_1, f'_2) \wedge E_v(l_1, l_2, l'_1, l'_2)$

$\alpha_a \equiv \exists v_1 v_2 w_1 w_2 v'_1 v'_2 w'_1 w'_2 p_{11} p_{12} p_{21} p_{22} p_{31} p_{32} p'_{11} p'_{12}\big($

$\text{A5.5} \wedge \text{A5.6} \wedge \text{A5.7} \wedge \text{A5.8} \wedge \text{A5.9}\big)$ where

- A5.5 expresses "$((v_1, v_2), (w_1, w_2))$ and $((v'_1, v'_2), (w'_1, w'_2))$ define the *window of change*, that is the segment of the formula that is affected (changed) in the transition from stage $x$ to stage $x+1$ of the evaluation (see Cases (a) and (b) in Figure 6.10)".
  $\text{A5.5} \equiv \text{PATH}_{E_C}(f_1, f_2, v_1, v_2) \wedge \text{PATH}_{E_C}(w_1, w_2, l_1, l_2) \wedge E_C(p_{11}, p_{12}, p_{21}, p_{22}) \wedge E_C(p_{21}, p_{22}, p_{31}, p_{32}) \wedge E_C(v_1, v_2, p_{11}, p_{12}) \wedge E_C(p_{31}, p_{32}, w_1, w_2) C_(v_1, v_2) \wedge C_)(w_1, w_2) \wedge \text{PATH}_{E_C}(f'_1, f'_2, v'_1, v'_2) \wedge \text{PATH}_{E_C}(w'_1, w'_2, l'_1, l'_2) \wedge E_C(v'_1, v'_2, p'_{11}, p'_{12}) \wedge E_C(p'_{11}, p'_{12}, w'_1, w'_2) \wedge E_v(p_{11}, p_{12}, p'_{11}, p'_{12}) \wedge E_v(v_1, v_2, v'_1, v'_2) \wedge E_v(w_1, w_2, w'_1, w'_2) \wedge C_(v'_1, v'_2) \wedge C_)(w'_1, w'_2)$

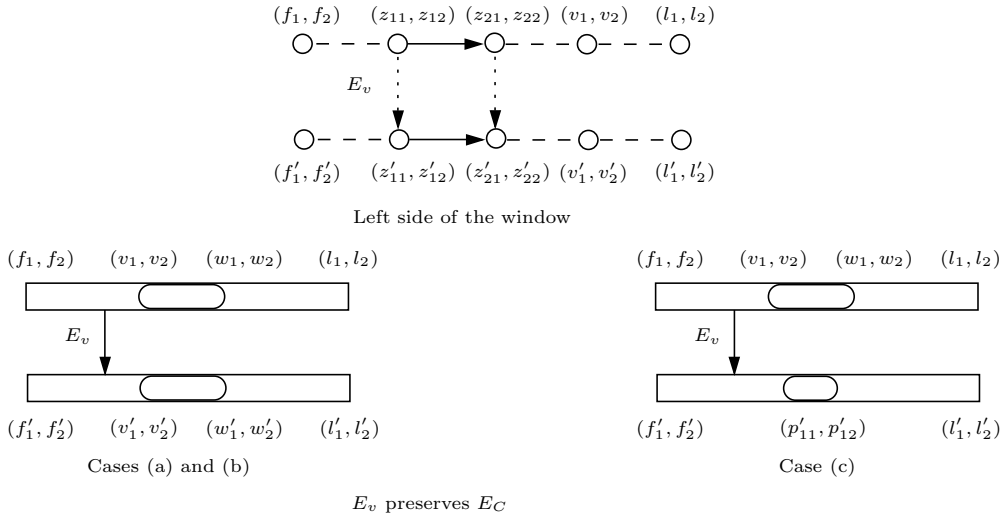


Figure 6.10

- A5.6 expresses "$E_v$ preserves $E_C$ outside of the window of change, and preserves left and right side of the window of change (see Figure 6.10)".

$$\begin{aligned}
\text{A5.6} \equiv \forall z_{11} z_{12} z_{21} z_{22} z'_{11} z'_{12} z'_{21} z'_{22} \big( & \\
& \big( (\text{PATH}_{E_C}(f_1, f_2, z_{11}, z_{12}) \wedge \text{PATH}_{E_C}(z_{21}, z_{22}, v_1, v_2) \wedge \\
& \quad E_C(z_{11}, z_{12}, z_{21}, z_{22}) \wedge E_v(z_{11}, z_{12}, z'_{11}, z'_{12}) \wedge E_v(z_{21}, z_{22}, z'_{21}, z'_{22})) \rightarrow \\
& \quad (\text{PATH}_{E_C}(f'_1, f'_2, z'_{11}, z'_{12}) \wedge \text{PATH}_{E_C}(z'_{21}, z'_{22}, v'_1, v'_2) \wedge \\
& \qquad E_C(z'_{11}, z'_{12}, z'_{21}, z'_{22}))) \wedge \\
& \big( (\text{PATH}_{E_C}(w_1, w_2, z_{11}, z_{12}) \wedge \text{PATH}_{E_C}(z_{21}, z_{22}, l_1, l_2) \wedge \\
& \quad E_C(z_{11}, z_{12}, z_{21}, z_{22}) \wedge E_v(z_{11}, z_{12}, z'_{11}, z'_{12}) \wedge E_v(z_{21}, z_{22}, z'_{21}, z'_{22})) \rightarrow \\
& \quad (\text{PATH}_{E_C}(w'_1, w'_2, z'_{11}, z'_{12}) \wedge \text{PATH}_{E_C}(z'_{21}, z'_{22}, l'_1, l'_2) \wedge \\
& \qquad E_C(z'_{11}, z'_{12}, z'_{21}, z'_{22})))\big)
\end{aligned}$$

- A5.7 expresses "$E_v$ preserves symbols in left side of the window of change".

$$\begin{aligned}
\text{A5.7} \equiv \forall z_{11} z_{12} z'_{11} z'_{12} \big( & \\
& \big( \text{PATH}_{E_C}(f_1, f_2, z_{11}, z_{12}) \wedge \text{PATH}_{E_C}(z_{11}, z_{12}, v_1, v_2) \wedge E_v(z_{11}, z_{12}, z'_{11}, z'_{12}) \big) \\
& \quad \rightarrow \big( \text{PATH}_{E_C}(f'_1, f'_2, z'_{11}, z'_{12}) \wedge \text{PATH}_{E_C}(z'_{11}, z'_{12}, v'_1, v'_2) \wedge \\
& \qquad \big( (C_(z_{11}, z_{12}) \wedge C_(z'_{11}, z'_{12})) \vee (C_)(z_{11}, z_{12}) \wedge C_)(z'_{11}, z'_{12})) \vee \\
& \qquad (C_\wedge(z_{11}, z_{12}) \wedge C_\wedge(z'_{11}, z'_{12})) \vee (C_\vee(z_{11}, z_{12}) \wedge C_\vee(z'_{11}, z'_{12})) \vee \\
& \qquad (C_0(z_{11}, z_{12}) \wedge C_0(z'_{11}, z'_{12})) \vee (C_1(z_{11}, z_{12}) \wedge C_1(z'_{11}, z'_{12})) \vee \\
& \qquad (C_\neg(z_{11}, z_{12}) \wedge C_\neg(z'_{11}, z'_{12})))))
\end{aligned}$$

- A5.8 expresses "$E_v$ preserves symbols in right side of the window of change".

$$\begin{aligned}
\text{A5.8} \equiv \forall z_{11} z_{12} z'_{11} z'_{12} \big( & \\
& \big( \text{PATH}_{E_C}(w_1, w_2, z_{11}, z_{12}) \wedge \text{PATH}_{E_C}(z_{11}, z_{12}, l_1, l_2) \wedge E_v(z_{11}, z_{12}, z'_{11}, z'_{12}) \big) \\
& \quad \rightarrow \big( \text{PATH}_{E_C}(w'_1, w'_2, z'_{11}, z'_{12}) \wedge \text{PATH}_{E_C}(z'_{11}, z'_{12}, l'_1, l'_2) \wedge \\
& \qquad \big( (C_(z_{11}, z_{12}) \wedge C_(z'_{11}, z'_{12})) \vee (C_)(z_{11}, z_{12}) \wedge C_)(z'_{11}, z'_{12})) \vee \\
& \qquad (C_\wedge(z_{11}, z_{12}) \wedge C_\wedge(z'_{11}, z'_{12})) \vee (C_\vee(z_{11}, z_{12}) \wedge C_\vee(z'_{11}, z'_{12})) \vee \\
& \qquad (C_0(z_{11}, z_{12}) \wedge C_0(z'_{11}, z'_{12})) \vee (C_1(z_{11}, z_{12}) \wedge C_1(z'_{11}, z'_{12})) \vee
\end{aligned}$$

$$(C_\neg(z_{11}, z_{12}) \wedge C_\neg(z'_{11}, z'_{12}))))\big)$$

- A5.9 expresses "In $(p'_{11}, p'_{12})$ we get the *result* of applying the operator $\theta$ in $(p_{21}, p_{22})$ to the Boolean values $b_1$, in $(p_{11}, p_{12})$, and $b_2$ in $(p_{31}, p_{32})$ (see (a) in Figure 6.9)".

$$\begin{aligned}\text{A5.9} \equiv \big(&(C_0(p_{11}, p_{12}) \wedge C_0(p_{31}, p_{32}) \wedge C_\wedge(p_{21}, p_{22}) \wedge C_0(p'_{11}, p'_{12}))\vee\\ &(C_0(p_{11}, p_{12}) \wedge C_0(p_{31}, p_{32}) \wedge C_\vee(p_{21}, p_{22}) \wedge C_0(p'_{11}, p'_{12}))\vee\\ &(C_0(p_{11}, p_{12}) \wedge C_1(p_{31}, p_{32}) \wedge C_\wedge(p_{21}, p_{22}) \wedge C_0(p'_{11}, p'_{12}))\vee\\ &(C_0(p_{11}, p_{12}) \wedge C_1(p_{31}, p_{32}) \wedge C_\vee(p_{21}, p_{22}) \wedge C_1(p'_{11}, p'_{12}))\vee\\ &(C_1(p_{11}, p_{12}) \wedge C_0(p_{31}, p_{32}) \wedge C_\wedge(p_{21}, p_{22}) \wedge C_0(p'_{11}, p'_{12}))\vee\\ &(C_1(p_{11}, p_{12}) \wedge C_0(p_{31}, p_{32}) \wedge C_\vee(p_{21}, p_{22}) \wedge C_1(p'_{11}, p'_{12}))\vee\\ &(C_1(p_{11}, p_{12}) \wedge C_1(p_{31}, p_{32}) \wedge C_\wedge(p_{21}, p_{22}) \wedge C_1(p'_{11}, p'_{12}))\vee\\ &(C_1(p_{11}, p_{12}) \wedge C_1(p_{31}, p_{32}) \wedge C_\vee(p_{21}, p_{22}) \wedge C_1(p'_{11}, p'_{12}))\big)\end{aligned}$$

The subformulae $\alpha_b$ and $\alpha_c$ that correspond to the cases $(b)$ and $(c)$ in Figure 6.9, are similar to $\alpha_a$. For the clarity of presentation, we omit those formulae. Furthermore, it should be clear how to build them using $\alpha_a$ as template. Moreover, the complete formulae can be found in [12]. We present next the remaining two subformulae, namely $\alpha_d$ and $\alpha_e$.

$$\begin{aligned}\alpha_d \equiv \exists y\big(&E_{ST}(x, y) \wedge \neg\exists z(E_{ST}(y, z))\wedge\\ &\exists p_{11}p_{12}p'_{11}p'_{12}\big(M(x, f_1, f_2) \wedge M(y, p'_{11}, p'_{12})\wedge\\ &\qquad E_C(f_1, f_2, p_{11}, p_{12}) \wedge E_C(p_{11}, p_{12}, l_1, l_2) \wedge E_C(l_1, l_2, p'_{11}, p'_{12})\wedge\\ &\qquad \neg\exists p'_{21}p'_{22}(E_C(p'_{11}, p'_{12}, p'_{21}, p'_{22}))\wedge\\ &\qquad C_(f_1, f_2) \wedge C_)(l_1, l_2)\wedge\\ &\qquad ((C_1(p_{11}, p_{12}) \wedge C_1(p'_{11}, p'_{12})) \vee (C_0(p_{11}, p_{12}) \wedge C_0(p'_{11}, p'_{12}))))\big)\end{aligned}$$

Note that the first line in $\alpha_d$ expresses "$x$ is the predecessor of the leaf in $E_{ST}$", so that this case corresponds to the last transition (see Figure 6.11). Also note that the last transition is necessarily an instance of case (c) in Figure 6.9"

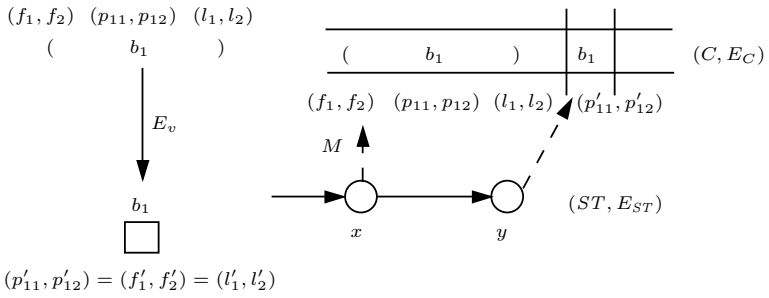


Figure 6.11

$\alpha_e \equiv$ A5.10 $\wedge\ \exists p'_1p'_2\big(M(x, p'_1, p'_2) \wedge$ A5.11 $\wedge$ A5.12$\big)$ where

- A5.10 expresses "$x$ is the leaf in $E_{ST}$".
  A5.10 $\equiv \neg\exists y(E_{ST}(x,y))$
- A5.11 expresses "$(p'_1,p'_2)$ is the leaf in $E_C$".
  A5.11 $\equiv \neg\exists y'_1y'_2(E_C(p'_1,p'_2,y'_1,y'_2))$
- A5.12 expresses "the last formula in $(C,E_C)$ is 1".
  A5.12 $\equiv C_1(p'_1,p'_2)$

## *6.3 Auxiliary Formulae*

For the sake of completeness, we define next the remaining auxiliary formulae used through the previous subsections. We assume an edge relation $E$ and a total order $\leq$.

"$x=0$" $\equiv \neg\exists y(y\neq x\wedge y\leq x)$

"$x=1$" $\equiv \exists y(y\neq x\wedge y\leq x\wedge\neg\exists z(z\neq x\wedge z\neq y\wedge y\leq z\wedge z\leq x)\wedge\neg\exists z(z\neq y\wedge z\leq y))$

$\mathrm{SUC}_{\leq}(x,y) \equiv x\leq y\wedge y\neq x\wedge\neg\exists z(z\neq x\wedge z\neq y\wedge x\leq z\wedge z\leq y)$

$\mathrm{PRED}_{\leq}(y,x) \equiv \mathrm{SUC}_{\leq}(x,y)$

$\mathrm{PATH}_{\mathrm{E}}(v,w)$ is used to denote the following formula which is satisfied by a given graph $\mathbf{G}$ iff $(v,w)$ is in the transitive closure of the relation $E^{\mathbf{G}}$.

$\mathrm{PATH}_E(v,w) \equiv v=w\vee\exists V'E'\big(V'(v)\wedge V'(w)\wedge \mathrm{A1}\wedge\mathrm{A2}\wedge\mathrm{A3}\wedge\mathrm{A4}\wedge\mathrm{A5}\big)$

- A1 expresses "$(V',E')$ is a subgraph of $(V,E)$ with no loops".
  A1 $\equiv \forall xy(E'(x,y)\rightarrow(V'(x)\wedge V'(y)\wedge E(x,y)))\wedge\forall x(V'(x)\rightarrow V(x))\wedge\forall x(\neg E'(x,x))$
- A2 expresses "$v$ is the only minimal node".
  A2 $\equiv \neg\exists x(E'(x,v))\wedge\forall y((V'(y)\wedge y\neq v)\rightarrow\exists x(E'(x,y)))$
- A3 expresses "$w$ is the only maximal node".
  A3 $\equiv \neg\exists x(E'(w,x))\wedge\forall y((V'(y)\wedge y\neq w)\rightarrow\exists x(E'(y,x)))$
- A4 expresses "all nodes except $v$ have input degree 1".
  A4 $\equiv \forall z((V'(z)\wedge z\neq v)\rightarrow\exists x(E'(x,z)\wedge\forall y((V'(y)\wedge E'(y,z))\rightarrow y=x)))$
- A5 expresses "all nodes except $w$ have output degree 1".
  A5 $\equiv \forall z((V'(z)\wedge z\neq w)\rightarrow\exists x(E'(z,x)\wedge\forall y((V'(y)\wedge E'(z,y))\rightarrow y=x)))$

That is, $\mathrm{PATH}_E(v,w)$ expresses "$(V',E')$ is a linear subgraph of $(V,E)$, with minimal node $v$ and maximal node $w$". We use a similar strategy to define the next auxiliary formula $\mathrm{LINEAR}(V,E)$ which expresses "$(V,E)$ is a linear graph".

$$
\begin{aligned}
\mathrm{LINEAR}(V,E) \equiv\ & \forall xy(\mathrm{PATH}_E(x,y)\vee\mathrm{PATH}_E(y,x))\wedge\\
& (\exists xy(x\neq y)\rightarrow\forall x(\neg E(x,x)))\wedge\\
& \exists vw\big(V(v)\wedge V(w)\wedge\\
& \quad \neg\exists x(E(x,v))\wedge\forall y((V(y)\wedge y\neq v)\rightarrow\exists x(E(x,y)))\wedge\\
& \quad \neg\exists x(E(w,x))\wedge\forall y((V(y)\wedge y\neq w)\rightarrow\exists x(E(y,x)))\wedge\\
& \quad \forall z((V(z)\wedge z\neq v)\rightarrow\\
& \quad\quad \exists x(E(x,z)\wedge\forall y((V(y)\wedge E(y,z))\rightarrow y=x)))\\
& \quad \forall z\big((V(z)\wedge z\neq w)\rightarrow\\
& \quad\quad \exists x(E(z,x)\wedge\forall y((V(y)\wedge E(z,y))\rightarrow y=x))\big)\big)
\end{aligned}
$$

Note that we only allow loops in a linear graph when it has only one node.

In a similar way we can define the second-order formula $\mathrm{LINEAR}_2(V, E)$ where the free second-order variables have arity 2 and 4 respectively.

We also use the formula $\mathrm{PATH}_{\mathrm{E_C}}(x_1, x_2, y_1, y_2)$ with free first-order variables $x_1$, $x_2$, $y_1$, $y_2$, where the set of vertices is a binary relation, and the set of edges is a 4-ary relation (see Figures 6.7 and 6.8).

## 7 SATQBF in Third-Order Logic

In this section we show how to build a formula in third-order logic that expresses SATQBF. We omit the tedious details of the subformulae which can be built following the same patterns than in the detailed exposition of the second-order formula for $\mathrm{SATQBF}_k$.

Roughly, we first express the existence of a third-order alternating valuation $\mathbf{T}_v$ applicable to a given QBF formula $\varphi$. Then we proceed to evaluate the quantifier-free part $\varphi'$ of $\varphi$ on each leaf valuation $\mathbf{L}_v$ of $\mathbf{T}_v$. For this part we use the same second-order subformulae than for $\mathrm{SATQBF}_k$. That is, from $\varphi'$ and $\mathbf{L}_v$, we build a Boolean sentence $\phi$ on $\{0, 1\}$ by replacing each occurrence of a Boolean variable $x$ in $\varphi'$ by a constant 0 or 1 according to the Boolean value assigned by $\mathbf{L}_v$ to $x$, and then we evaluate $\phi$.

Unlike the case with $\mathrm{SATQBF}_k$ in which the input formulae all have a same *fixed* number $k$ of alternating blocks of quantifiers, in the case of SATQBF the number of alternating blocks $k \geq 1$ of quantifiers in the input formulae is *not fixed*. That is, we need to take into account that the input formula can have any arbitrary number $k \geq 1$ of alternating blocks of quantifiers. We assume w.l.o.g. that the quantification in the input formula $\varphi$ has the form

$$\exists x_{11} \cdots \exists x_{1l_1} \forall x_{21} \cdots \forall x_{2l_2} \exists x_{31} \cdots \exists x_{3l_3} \cdots Q x_{k1} \cdots Q x_{kl_k} ( \\ \varphi'(x_{11}, \ldots, x_{1l_1}, x_{21} \ldots, x_{2l_2}, x_{31}, \ldots, x_{3l_3}, \ldots, x_{k1}, \ldots, x_{kl_k}))$$

where $k \geq 1$, the formula $\varphi'$ is a quantifier-free Boolean formula and $Q$ is $\exists$ if $k$ is odd, or $\forall$ if $k$ is even. To represent the formulae as relational structures, we use the same encoding based in word models as in Section 6.

We present a sketch of the third-order formula $\varphi_{\mathrm{SATQBF}}$ that expresses SATQBF. We follow a top-down approach, leaving most of the fine details of the formulae in the lowest level of abstraction as an exercise for the reader. At the highest level of abstraction, we can think of $\varphi_{\mathrm{SATQBF}}$ as a third-order formula that expresses.

"There is a third-order alternating valuation $\mathbf{T}_v$ applicable to $\varphi$, which satisfies $\varphi$".

At the next level of abstraction we can express $\varphi_{\mathrm{SATQBF}}$ in third-order logic as follows.

$\exists \mathcal{V}_t \, \mathcal{E}_t \, \mathcal{B}_t \, V_t \, E_t \, \Big( \mathrm{A1} \wedge \mathrm{A2} \wedge \mathrm{A3} \wedge \mathrm{A4} \wedge \mathrm{A5} \Big)$ where

- A1 expresses "$\mathcal{B}_t : \mathcal{V}_t \to \{0, 1\}$".
- A2 expresses "$G_t = (V_t, E_t)$ is a linear graph which represents the sequence of quantified variables in $\varphi$".
- A3 expresses "$(\mathcal{V}_t, \mathcal{E}_t)$ is a third-order binary tree with all its leaves at the same depth, which is in turn equal to the length of $(E_t, V_t)$".

- A4 expresses "$(\mathcal{V}_t, \mathcal{E}_t, \mathcal{B}_t)$ is a *third-order alternating valuation* $\mathbf{T}_v$ applicable to $\varphi$, i.e., all the nodes in $(\mathcal{V}_t, \mathcal{E}_t)$ whose depth correspond to a universally quantified variable in the prefix of quantifiers of $\varphi$, have exactly one sibling, and its value under $\mathcal{B}_t$ is different than that of the given node, and all the nodes whose depth correspond to an existentially quantified variable in the prefix of quantifiers of $\varphi$, are either the root or have no siblings".
- A5 expresses "Every *leaf valuation* in $(\mathcal{V}_t, \mathcal{E}_t, \mathcal{B}_t)$ satisfies $\varphi'$".

Recall that we use uppercase calligraphic letters for third-order variables and plain uppercase letters for second-order variables. In particular, $\mathcal{V}_t$, $\mathcal{E}_t$ and $\mathcal{B}_t$ are third-order variables while $V_t$ and $E_t$ are second-order variables.

Finally, we describe the strategies to express A2–A5 in third-order logic.

A2. $\text{LINEAR}(V_t, E_t) \wedge$ A2.1 where
- A2.1 expresses "The length of $G_t$ is equal to the number of variables in the prefix of quantifiers of $\varphi$. That is, there is a relation $V_p$ which is a partial bijection from the quantifier prefix of $\varphi$ (restricted to the $X$'s in the quantifier prefix) to $V_t$, which maps every $X$ in the quantifier prefix to its corresponding node in $G_t$, and which preserves $E_t$ and $\leq^{\mathbf{G}_\varphi}$ in $G_t$ and $\varphi$ (restricted to the $X$'s in the quantifier prefix), respectively".
  See (A) in Subsection 6.2.1 for more details.

A3. Let $\mathcal{E}_t \restriction_{\mathcal{S}_d}$ denote the restriction of the third-order relation $\mathcal{E}_t$ to the nodes in the third-order set $\mathcal{S}_d$. We can express A3 as follows:
A3.1 $\wedge$ A3.2 $\wedge$ A3.3 $\wedge$ A3.4 where
- A3.1 expresses "$(\mathcal{V}_t, \mathcal{E}_t)$ is a third-order connected graph that has one root and one or more leaves".
- A3.2 expresses "Except for the root node, all nodes in $(\mathcal{V}_t, \mathcal{E}_t)$ have input degree 1".
- A3.3 expresses "Except for the leaf nodes, all nodes in $(\mathcal{V}_t, \mathcal{E}_t)$ have output degree 1 or 2".
- A3.4 expresses "All leaf nodes in $(\mathcal{V}_t, \mathcal{E}_t)$ have the same depth, which is in turn equal to the length of $(V_t, E_t)$"

A3.1, A3.2 and A3.3 can be expressed in third-order logic as follows:

$$\begin{aligned}
&\exists R\big(\forall Z(\mathcal{V}_t(Z) \rightarrow \text{PATH}_{\mathcal{E}_t}(R, Z)) \wedge \\
&\quad \neg\exists S_1(\mathcal{E}_t(S_1, R)) \wedge \\
&\quad \exists S_1(\neg\exists S_2(\mathcal{E}_t(S_1, S_2))) \wedge \\
&\quad \forall Z((\mathcal{V}_t(Z) \wedge Z \neq R) \rightarrow \exists S_1(\mathcal{E}_t(S_1, Z) \wedge \forall S_2(\mathcal{E}_t(S_2, Z) \rightarrow S_1 = S_2)))\big) \wedge \\
&\forall Z\big(\mathcal{V}_t(Z) \rightarrow \neg\exists S_1 S_2 S_3(S_1 \neq S_2 \wedge S_2 \neq S_3 \wedge S_1 \neq S_3 \wedge \mathcal{E}_t(Z, S_1) \wedge \mathcal{E}_t(Z, S_2) \wedge \\
&\qquad\qquad \mathcal{E}_t(Z, S_3))\big)
\end{aligned}$$

Regarding A3.4, we can express it as follows:
$\forall X\big(\text{A3.4.1} \rightarrow \big(\exists \mathcal{S}_d\, \mathcal{D}(\text{A3.4.2} \wedge \text{A3.4.3} \wedge \mathcal{S}_d(X) \wedge \text{A3.4.4} \wedge \text{A3.4.5})\big)\big)$ where
- A3.4.1 expresses "$X$ is a leaf node in $(\mathcal{V}_t, \mathcal{E}_t)$".
- A3.4.2 expresses "$\mathcal{S}_d \subseteq \mathcal{V}_t$".
- A3.4.3 expresses "$\mathcal{D} : V_t \rightarrow \mathcal{S}_d$ is a bijection that preserves $E_t$ and $\mathcal{E}_t \restriction_{\mathcal{S}_d}$".
- A3.4.4 expresses "$\mathcal{D}^{-1}(X)$ is the leaf node in $G_t = (V_t, E_t)$".
- A3.4.5 expresses "$\mathcal{S}_d$ includes the root of $(\mathcal{V}_t, \mathcal{E}_t)$".

A4. We can express A4 as follows (refer to Figures 6.2 and 6.5):
A4.1 $\wedge\, \forall x \forall \mathcal{S}_d\big((V_t(x) \wedge \text{A4.2} \wedge \text{A4.3}) \rightarrow$
$\forall \mathcal{D}((\text{A4.4} \wedge \text{A4.5}) \rightarrow ((\text{A4.6} \rightarrow \text{A4.7}) \wedge (\text{A4.8} \rightarrow \text{A4.9})))\big)$ where
– A4.1 expresses "$\mathcal{B}_t$ is a total function from $\mathcal{V}_t$ to $\{0,1\}$".
– A4.2 expresses "$\mathcal{S}_d \subseteq \mathcal{V}_t$".
– A4.3 expresses "$(\mathcal{S}_d, \mathcal{E}_t \restriction_{\mathcal{S}_d})$ is a linear graph which includes the root of $(\mathcal{V}_t, \mathcal{E}_t)$".
– A4.4 expresses "$\mathcal{D}$ is a bijection from the initial subgraph of $G_t$ up to $x$, to $\mathcal{S}_d$".
– A4.5 expresses "$\mathcal{D}$ preserves $E_t$ and $\mathcal{E}_t \restriction_{\mathcal{S}_d}$".
– A4.6 expresses "the predecessor of $V_p^{-1}(x)$ in $\leq^{\mathbf{G}_\varphi}$ is $\forall$".
– A4.7 expresses "$\mathcal{D}(x)$ has exactly one sibling in $(\mathcal{V}_t, \mathcal{E}_t)$ and $\mathcal{B}_t$ of that sibling is not equal to $\mathcal{B}_t(\mathcal{D}(x))$".
– A4.8 expresses "the predecessor of $V_p^{-1}(x)$ in $\leq^{\mathbf{G}_\varphi}$ is $\exists$".
– A4.9 expresses "$\mathcal{D}(x)$ has no siblings in $(\mathcal{V}_t, \mathcal{E}_t)$, or $\mathcal{D}(x)$ is the root in $(\mathcal{V}_t, \mathcal{E}_t)$".

A5. $\forall \mathcal{S}_v\big((\text{A5.1} \wedge \text{A5.2}) \rightarrow \exists \mathcal{D}\, B_t\big(\text{A5.3} \wedge \text{A5.4} \wedge \text{A5.5}\big)\big)$ where
– A5.1 expresses "$\mathcal{S}_v \subseteq \mathcal{V}_t$".
– A5.2 expresses "$(\mathcal{S}_v, \mathcal{E}_t \restriction_{\mathcal{S}_v})$ is a linear graph which includes the root and a leaf of $(\mathcal{V}_t, \mathcal{E}_t)$".
– A5.3 expresses "$\mathcal{D}$ is a bijection from $V_t$ to $\mathcal{S}_v$ which preserves $E_t$ and $\mathcal{E}_t \restriction_{\mathcal{S}_v}$".
– A5.4 expresses "$B_t$ is a total function from $V_t$ to $\{0,1\}$ which coincides with $\mathcal{B}_t(\mathcal{S}_v)$ w.r.t. $\mathcal{D}$".
– A5.5 expresses "the leaf valuation represented by $(V_t, E_t, B_t)$ satisfies the quantifier-free subformula $\varphi'$ of $\varphi$".

Note that, A5.5 can be expressed as in Subsection 6.2.2.

Remark 7.1
Note that while in the third-order formulae in A4 and A5 we have used universal third-order quantification (for $\mathcal{S}_d$ and $\mathcal{D}$ in A4, and for $\mathcal{S}_v$ in A5), it is not actually needed, and existential third-order quantification is enough. These are the only subformulae where we have used universal third-order quantification. Hence, we strongly believe that our third-order formula can be translated in a rather technical way into an existential third-order formula.

Let us consider the sketch for an existential third-order formula equivalent to the formula in A4 (the existential formula for A5 is easier). We can say that for every node $x$ in the graph $(V_t, E_t)$, and for every set $Z$ that is a node in the third-order graph $(\mathcal{V}_t, \mathcal{E}_t)$, and such that there is a third-order set $\mathcal{S}_d$ of nodes in the third-order graph $(\mathcal{V}_t, \mathcal{E}_t)$, such that the restriction of the edge relation $\mathcal{E}_t$ to the third-order set $\mathcal{S}_d$, together with $\mathcal{S}_d$, form a (third-order) subgraph that is a linear graph whose root is the root of the third-order graph $(\mathcal{V}_t, \mathcal{E}_t)$, and whose leaf is the set $Z$, and such that its length is the length of the initial subgraph of the graph $(V_t, E_t)$, up to the node $x$, if the variable represented by $x$ in the input formula $\varphi$ is universally quantified, then the node $Z$ in the third-order graph $(\mathcal{V}_t, \mathcal{E}_t)$ has exactly one sibling in that graph, and that sibling has a different value assigned by $\mathcal{B}_t$ than the value assigned by $\mathcal{B}_t$ to $Z$. On the other hand, if the variable represented by $x$ in the input formula $\varphi$ is existentially quantified, then the node $Z$ in the third-order graph $(\mathcal{V}_t, \mathcal{E}_t)$ has no sibling in that graph. To say that "the third order graph induced by the set $\mathcal{S}_d$ in the graph $(\mathcal{V}_t, \mathcal{E}_t)$, whose leaf is the set $Z$, has the same length as the initial subgraph of the graph $(V_t, E_t)$, up to the node $x$", we say that there is a binary third-order

relation $\mathcal{D}$ which is a bijection between the set of nodes in the initial subgraph of the graph $(V_t, E_t)$, up to the node $x$, and the third-order set $\mathcal{S}_d$, and which preserves $E_t$ and the restriction of the edge relation $\mathcal{E}_t$ to the third-order set $\mathcal{S}_d$.

## 8 Final Considerations

Let $\exists\mathrm{SO}^{\leq 2}$ denote the restriction of $\exists$SO to formulae with second-order variables of arity $\leq 2$. As pointed out in [2], it is open whether on graphs full $\exists$SO is strictly more expressive than $\exists\mathrm{SO}^{\leq 2}$. Also as pointed out in [2], no concrete example of a graph property in PSPACE that is not in binary NP has been found yet, even though it is known that such properties exist. Hence, it would be worthwhile to find an example of a PSPACE query on graphs that cannot be expressed in $\exists\mathrm{SO}^{\leq 2}$. The gained experience on writing non-trivial queries in second-order logic, can prove to be a valuable platform to make progress on these kind of open problems. In particular, we used a second-order variable of arity 4 in Section 6. We used it to represent (together with other variables) a linear digraph which, for each of the leaf valuations, encodes a sequence of word models corresponding to the different stages of evaluation of the quantifier free part of the input $\mathrm{QBF}_k$ formula. Since the size of the Boolean formula in each stage is linear in the size of the input $\mathrm{QBF}_k$ formula, and the number of connectives in the formula is also linear, the length of the complete sequence of Boolean formulae is quadratic. Therefore, we conjecture that arity 4 is actually a lower bound, though we have not attempted to prove it yet. In general, the exploration of properties which force us to work with intermediate structures of size greater than linear w.r.t. the input, seems a reasonable way of approaching these kind of open problems.

As noted earlier, there are second-order queries that are difficult to express in the language of second-order logic, but which have an elegant and simple characterization in third-order logic. Therefore it would be interesting to explore possible characterizations of fragments of third-order logic that admit translations of their formulae to equivalent formulae in second-order logic. This way, those fragments of third-order logic could be assimilated to high-level programming languages, while second-order logic would be the corresponding low-level programming language. In turn, this would allow us to express complex second-order queries with greater abstraction of the low-level details, thus minimizing the probability of error.

## References

[1] José Luis Balcázar, Joseph Díaz, and Joaquim Gabarró. *Structural Complexity I.* Texts in Theoretical Computer Science, EATCS. Springer, Berlin Heidelberg New York, 2 edition, 1995.

[2] Arnaud Durand, Clemens Lautemann, and Thomas Schwentick. Subclasses of binary NP. *J. Log. Comput.*, 8(2):189–207, 1998.

[3] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory.* Perspectives in Mathematical Logic. Springer, Berlin Heidelberg New York, 2nd edition, 1999.

[4] Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. Karp, editor, *Complexity of Computations*, volume 7 of *SIAM-AMS Proc.*, pages 27–41, Providence, RI, 1974. American Mathematical Society.

[5] Flavio Antonio Ferrarotti. *Expressibility of Higher-Order Logics on Relational Databases: Proper Hierarchies.* PhD thesis, Department of Information Systems, Massey University, Wellington, New Zealand, June 2008.

[6] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[7] Lauri Hella and José María Turull Torres. Computing queries with higher-order logics. *Theor. Comput. Sci.*, 355(2):197–214, 2006.

[8] Neil Immerman. *Descriptive Complexity*. Graduate Texts in Computer Science. Springer, Berlin Heidelberg New York, 1999.

[9] Michal Krynicki and Jose Maria Turull Torres. Games on trees and syntactical complexity of formulas. *Logic Journal of the IGPL*, 15(5-6):653–687, 2007.

[10] Leonid Libkin. *Elements Of Finite Model Theory*. Texts in Theoretical Computer Science, EATCS. Springer, Berlin Heidelberg New York, 2004.

[11] Wei Ren. Logic languages: Cubic graphs in second-order logic, 2008. Research Report, Massey University, Directed by José María Turull Torres.

[12] Wei Ren. On the descriptive complexity of satisfability on quantified boolean formulas. `http://mro.massey.ac.nz/handle/10179/2629`, 2011. Master of Information Sciences Thesis, Massey University, Directed by José María Turull Torres.

[13] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

[14] Celia Wrathall. Complete sets and the polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):23–33, 1976.

Received 21 December 2012.