Church's Type Theory
(Stanford Encyclopedia of Philosophy)

Peter B. Andrews Christoph Benzmüller

Church's Type Theory

First published Fri Aug 25, 2006; substantive revision Fri Feb 14, 2014; substantive revision Mon Jan 7, 2019

Church's type theory, aka simple type theory, is a formal logical language which includes classical first-order and propositional logic, but is more expressive in a practical sense. It is used, with some modifications and enhancements, in most modern applications of type theory. It is particularly well suited to the formalization of mathematics and other disciplines and to specifying and verifying hardware and software. It also plays an important role in the study of the formal semantics of natural language. When utilizing it as a meta-logic to semantically embed expressive (quantified) non-classical logics further topical applications are enabled in artificial intelligence and philosophy.

A great wealth of technical knowledge can be expressed very naturally in it. With possible enhancements, Church's type theory constitutes an excellent formal language for representing the knowledge in automated information systems, sophisticated automated reasoning systems, systems for verifying the correctness of mathematical proofs, and a range of projects involving logic and artificial intelligence. Some examples and further references are given in Sections 1.2.2 and 5 below.

Type theories are also called higher-order logics, since they allow quantification not only over individual variables (as in first-order logic), but also over function, predicate, and even higher order variables. Type theories characteristically assign types to entities, distinguishing, for example, between numbers, sets of numbers, functions from numbers to sets of numbers, and sets of such functions. As illustrated in Section 1.2.2 below, these distinctions allow one to discuss the conceptually rich world of sets and functions without encountering the paradoxes of naive set theory.

Church's type theory is a formulation of type theory that was introduced by Alonzo Church in Church 1940. In certain respects, it is simpler and more general than the type theory introduced by Bertrand Russell in Russell 1908 and Whitehead & Russell 1927a. Since properties and relations can be regarded as functions from entities to truth values, the concept of a function is taken

as primitive in Church's type theory, and the λ-notation which Church introduced in Church 1932 and Church 1941 is incorporated into the formal language. Moreover, quantifiers and description operators are introduced in way so that additional binding mechanisms can be avoided, λ-notation is reused instead. λ-notation is thus the only binding mechanism employed in Church's type theory.

Contents

1 Syntax

1.1 Fundamental Ideas

We start with an informal description of the fundamental ideas underlying the syntax of Church's formulation of type theory.

All entities have types, and if α and β are types, the type of functions from elements of type β to elements of type α is written as $(\alpha\beta)$. (This notation was introduced by Church, but some authors write $(\beta \to \alpha)$ instead of $(\alpha\beta)$. See, for example, Section 2 of the entry on type theory.)

As noted by Schönfinkel (1924), functions of more than one argument can be represented in terms of functions of one argument when the values of these functions can themselves be functions. For example, if f is a function of two arguments, for each element x of the left domain of f there is a function g (depending on x) such that $gy = fxy$ for each element y of the right domain of f. We may now write $g = fx$, and regard f as a function of a single argument, whose value for any argument x in its domain is a function fx, whose value for any argument y in its domain is fxy.

For a more explicit example, consider the function $+$ which carries any pair of natural numbers to their sum. We may denote this function by $+_{((\sigma\sigma)\sigma)}$, where σ is the type of natural numbers. Given any number x, $[+_{((\sigma\sigma)\sigma)}x]$ is the function which, when applied to any number y, gives the value $[[+_{((\sigma\sigma)\sigma)}x]y]$, which is ordinarily abbreviated as $x + y$. Thus $[+_{((\sigma\sigma)\sigma)}x]$ is the function of one argument which adds x to any number. When we think of $+_{((\sigma\sigma)\sigma)}$ as a function of one argument, we see that it maps any number x to the function $[+_{((\sigma\sigma)\sigma)}x]$.

More generally, if f is a function which maps n-tuples $\langle w_\beta, x_\gamma, \ldots, y_\delta, z_\tau\rangle$ of elements of types $\beta, \gamma, \ldots, \delta, \tau$, respectively, to elements of type α, we may assign to f the type $((\ldots((\alpha\tau)\delta)\ldots\gamma)\beta)$. It is customary to use the convention of association to the left to omit parentheses, and write this type symbol simply as $(\alpha\tau\delta\ldots\gamma\beta)$.

A set or property can be represented by a function (often called *characteristic function*) which maps elements to truth values, so that an element is in the set, or has the property, in question iff the function representing the set or property maps that element to truth. When a statement is asserted, the speaker means that it is true, so that sx means that sx is true, which also expresses

the assertions that s maps x to truth and that $x \in s$. In other words, $x \in s$ iff sx. We take o as the type symbol denoting the type of truth values, so we may speak of any function of type $(o\alpha)$ as a *set* of elements of type α. A function of type $((o\alpha)\beta)$ is a binary relation between elements of type β and elements of type α. For example, if σ is the type of the natural numbers, and $<$ is the order relation between natural numbers, $<$ has type $(o\sigma\sigma)$, and for all natural numbers x and $y, <xy$ (which we ordinarily write as $x < y$) has the value truth iff x is less than y. Of course, $<$ can also be regarded as the function which maps each natural number x to the set $<x$ of all natural numbers y such that x is less than y. Thus sets, properties, and relations may be regarded as particular kinds of functions. Church's type type theory is thus a logic of functions, and, in this sense, it is in the tradition of the work of Frege's Begriffsschrift. The opposite approach would be to reduce functions to relations, which was the approach taken by Whitehead and Russell (1927a) in the Principia Mathematica.

Expressions which denote elements of type α are called *wffs of type* α. Thus, statements of type theory are wffs of type o.

If $\mathbf{A}_\alpha$ is a wff of type α in which $\mathbf{u}_{\alpha\beta}$ is not free, the function (associated with) $\mathbf{u}_{\alpha\beta}$ such that $\forall \mathbf{v}_\beta[\mathbf{u}_{\alpha\beta}\mathbf{v}_\beta = \mathbf{A}_\alpha]$ is denoted by $[\lambda \mathbf{v}_\beta \mathbf{A}_\alpha]$. Thus $\lambda \mathbf{v}_\beta$ is a variable-binder, like $\forall \mathbf{v}_\beta$ or $\exists \mathbf{v}_\beta$ (but with a quite different meaning, of course); λ is known as an *abstraction operator*. $[\lambda \mathbf{v}_\beta \mathbf{A}_\alpha]$ denotes the function whose value on any argument $\mathbf{v}_\beta$ is $\mathbf{A}_\alpha$, where $\mathbf{v}_\beta$ may occur free in $\mathbf{A}_\alpha$. For example, $[\lambda n_\sigma[4 \cdot n_\sigma + 3]]$ denotes the function whose value on any natural number n is $4 \cdot n + 3$. Hence when we apply this function to the number 5 we obtain $[\lambda n_\sigma[4 \cdot n_\sigma + 3]]5 = 4 \cdot 5 + 3 = 23$.

We use $\mathsf{Sub}(\mathbf{B}, \mathbf{v}, \mathbf{A})$ as a notation for the result of substituting $\mathbf{B}$ for $\mathbf{v}$ in $\mathbf{A}$, and $\mathsf{SubFree}(\mathbf{B}, \mathbf{v}, \mathbf{A})$ as a notation for the result of substituting $\mathbf{B}$ for all free occurrences of $\mathbf{v}$ in $\mathbf{A}$. The process of replacing $[\lambda \mathbf{v}_\beta \mathbf{A}_\alpha]\mathbf{B}_\beta$ by $\mathsf{SubFree}(\mathbf{B}_\beta, \mathbf{v}_\beta, \mathbf{A}_\alpha)$ (or vice-versa) is known as *β-conversion*, which is one form of *λ-conversion*. Of course, when $\mathbf{A}_o$ is a wff of type o, $[\lambda \mathbf{v}_\beta \mathbf{A}_o]$ denotes the set of all elements $\mathbf{v}_\beta$ (of type β) of which $\mathbf{A}_o$ is true; this set may also be denoted by $\{\mathbf{v}_\beta | \mathbf{A}_o\}$. For example, $[\lambda x\ x < y]$ denotes the set of x such that x is less than y (as well as that property which a number x has if it is less than y. In familiar set-theoretic notation, $[\lambda \mathbf{v}_\beta \mathbf{A}_o]\mathbf{B}_\beta = \mathsf{SubFree}(\mathbf{B}_\beta, \mathbf{v}_\beta, \mathbf{A}_o)$ would be written $\mathbf{B}_\beta \in \{\mathbf{v}_\beta | \mathbf{A}_o\} \equiv \mathsf{SubFree}(\mathbf{B}_\beta, \mathbf{v}_\beta, \mathbf{A}_o)$. (By the Axiom of Extensionality for truth values, when $\mathbf{C}_o$ and $\mathbf{D}_o$ are of type $o, \mathbf{C}_o \equiv \mathbf{D}_o$ is equivalent to $\mathbf{C}_o = \mathbf{D}_o$.)

Propositional connectives and quantifiers can be assigned types and can be denoted by constants of these types. The negation function maps truth values to truth values, so it has type (oo). Similarly, disjunction and conjunction (etc.) are binary functions from truth values to truth values, so they have type (ooo).

The statement $\forall \mathbf{x}_\alpha \mathbf{A}_o$ is true iff the set $[\lambda \mathbf{x}_\alpha \mathbf{A}_o]$ contains all elements of type α. A constant $\Pi_{o(o\alpha)}$ can be introduced (for each type symbol α) to denote a property of sets: a set $s_{o\alpha}$ has the property $\Pi_{o(o\alpha)}$ iff $s_{o\alpha}$ contains all elements of type α. With this interpretation

$$\forall s_{o\alpha}[\Pi_{o(o\alpha)} s_{o\alpha} \equiv \forall x_\alpha[s_{o\alpha} x_\alpha]]$$

should be true, as well as

$$\Pi_{o(o\alpha)}[\lambda \mathbf{x}_\alpha \mathbf{A}_o] \equiv \forall \mathbf{x}_\alpha [[\lambda \mathbf{x}_\alpha \mathbf{A}_o]\mathbf{x}_\alpha] \tag{1}$$

for any wff $\mathbf{A}_o$ and variable $\mathbf{x}_\alpha$. Since by λ-conversion we have

$$[\lambda \mathbf{x}_\alpha \mathbf{A}_o]\mathbf{x}_\alpha \equiv \mathbf{A}_o$$

equation (1) can be written more simply as

$$\Pi_{o(o\alpha)}[\lambda \mathbf{x}_\alpha \mathbf{A}_o] \equiv \forall \mathbf{x}_\alpha \mathbf{A}_o$$

Thus, $\forall \mathbf{x}_\alpha$ can be defined in terms of $\Pi_{o(o\alpha)}$, and λ is the only variable-binder that is needed.

1.2 Formulas

Before we state the definition of a "formula", a word of caution is in order. The reader may be accustomed to thinking of a formula as an expression which plays the role of an assertion in a formal language, and of a term as an expression which designates an object. Church's terminology is somewhat different, and provides a uniform way of discussing expressions of many different types. What we call *well-formed formula* of type α (wff_α) below would in more standard terminology be called *term* of type α, and then only certain terms, namely those with type o, would be called formulas. Anyhow, in this entry we have decided to stay with Church's original terminology. Another remark concerns the use of some specific mathematical notation. In what follows, the entry distinguishes between the symbols $\imath, \iota$, and η. The first is the symbol used for the type of individuals; the second is the symbol used for a logical constant (see Section 1.2.1 below); the third is the symbol used as a variable-binding operator that represents the definite description "the" (see Section 1.3.4). The reader should not confuse them and check to see that the browser is displaying these symbols correctly.

1.2.1 Definitions

Type symbols are defined inductively as follows:

- $\imath$ is a type symbol (denoting the type of individuals). There may also be additional primitive type symbols which are used in formalizing disciplines where it is natural to have several sorts of individuals.
- o is a type symbol (denoting the type of truth values).
- If α and β are type symbols, then $(\alpha\beta)$ is a type symbol (denoting the type of functions from elements of type β to elements of type α).

The *primitive symbols* are the following:

- Improper symbols: [,], λ
- For each type symbol α, a denumerable list of *variables* of type α : $a_\alpha, b_\alpha, c_\alpha \ldots$
- Logical constants: $\sim_{(oo)}, \vee_{((oo)o)}, \Pi_{(o(o\alpha))}$ and $\iota_{(\alpha(o\alpha))}$ (for each type symbol α). The types of these constants are indicated by their subscripts. The symbol $\sim_{(oo)}$ denotes negation, $\vee_{((oo)o)}$ denotes disjunction, and $\Pi_{(o(o\alpha))}$ is used in defining the universal quantifier as discussed above. $\iota_{(\alpha(o\alpha))}$ serves either as a description or selection operator as discussed in Section 1.3.4 and Section 1.3.5 below.
- In addition, there may be other constants of various types, which will be called *nonlogical constants* or *parameters*. Each choice of parameters determines a particular formulation of the system of type theory. Parameters are typically used as names for particular entities in the discipline being formalized.

A *formula* is a finite sequence of primitive symbols. Certain formulas are called *well-formed formulas* (*wff*s). We write wff_α as an abbreviation for *wff of type* α, and define this concept inductively as follows:

- A primitive variable or constant of type α is a wff_α.
- If $\mathbf{A}_{\alpha\beta}$ and $\mathbf{B}_\beta$ are wffs of the indicated types, then $[\mathbf{A}_{\alpha\beta}\mathbf{B}_\beta]$ is a wff_α.
- If $\mathbf{x}_\beta$ is a variable of type β and $\mathbf{A}_\alpha$ is a wff, then $[\lambda\mathbf{x}_\beta\mathbf{A}_\alpha]$ is a $\text{wff}_{(\alpha\beta)}$.

Note, for example, that by (a) $\sim_{(oo)}$ is a $\text{wff}_{(oo)}$, so by (b) if $\mathbf{A}_o$ is a wff_o, then $[\sim_{(oo)} \mathbf{A}_o]$ is a wff_o. Usually, the latter wff will simply be written as $\sim \mathbf{A}$. It is often convenient to avoid parentheses, brackets and type symbols, and use conventions for omitting them. For formulas we use the convention of association to the right, and we may write $\vee_{((oo)o)}\mathbf{A}_o\mathbf{B}_o$ instead of $[[\vee_{((oo)o)}\mathbf{A}_o]\mathbf{B}_o]$. For types the corresponding convention is association to the left, and we may write ooo instead of $((oo)o)$.

Abbreviations:

- $\mathbf{A}_o \vee \mathbf{B}_o$ stands for $\vee_{ooo}\mathbf{A}_o\mathbf{B}_o$.
- $\mathbf{A}_o \supset \mathbf{B}_o$ stands for $[\sim_{oo} \mathbf{A}_o] \vee \mathbf{B}_o$.
- $\forall\mathbf{x}_\alpha\mathbf{A}_o$ stands for $\Pi_{o(o\alpha)}[\lambda\mathbf{x}_\alpha\mathbf{A}_o]$.
- Other propositional connectives, and the existential quantifier, are defined in familiar ways. In particular,

$$\mathbf{A}_o \equiv \mathbf{B}_o \quad \text{stands for} \quad [\mathbf{A}_o \supset \mathbf{B}_o] \wedge [\mathbf{B}_o \supset \mathbf{A}_o]$$

- $\mathsf{Q}_{o\alpha\alpha}$ stands for $\lambda\mathrm{x}_\alpha\lambda\mathrm{y}_\alpha\forall f_{o\alpha}[f_{o\alpha}x_\alpha \supset f_{o\alpha}y_\alpha]$.

- $\mathbf{A}_\alpha = \mathbf{B}_\alpha$ stands for $\mathsf{Q}_{o\alpha\alpha}\mathbf{A}_\alpha\mathbf{B}_\alpha$.

The last definition is known as the Leibnizian definition of equality. It asserts that x and y are the same if y has every property that x has. Actually, Leibniz called his definition "the identity of indiscernibles" and gave it in the form of a biconditional: x and y are the same if x and y have exactly the same properties. It is not difficult to show that these two forms of the definition are logically equivalent.

1.2.2 Examples

We now provide a few examples to illustrate how various assertions and concepts can be expressed in Church's type theory.

Example 1 To express the assertion that *"Napoleon is charismatic"* we introduce constants $Charismatic_{o\iota}$ and $Napoleon_\iota$, with the types indicated by their subscripts and the obvious meanings, and assert the wff

$$Charismatic_{o\iota}\, Napoleon_\iota$$

If we wish to express the assertion that *"Napoleon has all the properties of a great general"*, we might consider interpreting this to mean that *"Napoleon has all the properties of some great general"*, but it seems more appropriate to interpret this statement as meaning that *"Napoleon has all the properties which all great generals have"*. If the constant $GreatGeneral_{o\iota}$ is added to the formal language, this can be expressed by the wff

$$\forall p_{o\iota}[\forall g_\iota[GreatGeneral_{o\iota}g_\iota \supset p_{o\iota}g_\iota] \supset p_{o\iota}Napoleon_\iota]$$

As an example of such a property, we note that the sentence *"Napoleon's soldiers admire him"* can be expressed in a similar way by the wff

$$\forall x_\iota[Soldier_{o\iota}x_\iota \wedge CommanderOf_{\iota\iota}x_\iota = Napoleon_\iota \supset Admires_{o\iota\iota}x_\iota Napoleon_\iota]$$

By λ-conversion, this is equivalent to

$$[\lambda n_\iota \forall x_\iota[Soldier_{o\iota}x_\iota \wedge CommanderOf_{\iota\iota}x_\iota = n_\iota \supset Admires_{o\iota\iota}x_\iota n_\iota]]\, Napoleon_\iota$$

This statement asserts that one of the properties which Napoleon has is that of being admired by his soldiers. The property itself is expressed by the wff

$$\lambda n_\iota \forall x_\iota[Soldier_{o\iota}x_\iota \wedge CommanderOf_{\iota\iota}x_\iota = n_\iota \supset Admires_{o\iota\iota}x_\iota n_\iota]$$

Example 2 We illustrate some potential applications of type theory with the following fable.

"A rich and somewhat eccentric lady named Sheila has an ostrich and a cheetah as pets, and she wishes to take them from her hotel to her remote and almost

inaccessible farm. Various portions of the trip may involve using elevators, boxcars, airplanes, trucks, very small boats, donkey carts, suspension bridges, etc., and she and the pets will not always be together. She knows that she must not permit the ostrich and the cheetah to be together when she is not with them."

We consider how certain aspects of this problem can be formalized so that Sheila can use an automated reasoning system to help analyze the possibilities.

There will be a set *Moments* of instants or intervals of time during the trip. She will start the trip at the location *Hotel* and moment *Start*, and end it at the location *Farm* and moment *Finish*. Moments will have type τ, and locations will have type ϱ. A *state* will have type σ and will specify the location of Sheila, the ostrich, and the cheetah at a given moment. A *plan* will specify where the entities will be at each moment according to this plan. It will be a function from moments to states, and will have type $(\sigma\tau)$. The exact representation of states need not concern us, but there will be functions from states to locations called *LocationOfSheila*, *LocationOfOstrich*, and *LocationOfCheetah* which provide the indicated information. Thus, $LocationOfSheila_{\varrho\sigma}[p_{\sigma\tau}t_\tau]$ will be the location of Sheila according to plan $p_{\sigma\tau}$ at moment t_τ. The set $Proposals_{o(\sigma\tau)}$ is the set of plans Sheila is considering.

We define a plan p to be acceptable if, according to that plan, the group starts at the hotel, finishes at the farm, and whenever the ostrich and the cheetah are together, Sheila is there too. Formally, we define $Acceptable_{o(\sigma\tau)}$ as

$$\begin{aligned}
\lambda p_{\sigma\tau}\ [&LocationOfSheila_{\varrho\sigma}[p_{\sigma\tau}Start_\tau] = Hotel_\varrho \wedge \\
&LocationOfOstrich_{\varrho\sigma}[p_{\sigma\tau}Start_\tau] = Hotel_\varrho \wedge \\
&LocationOfCheetah_{\varrho\sigma}[p_{\sigma\tau}Start_\tau] = Hotel_\varrho \wedge \\
&LocationOfSheila_{\varrho\sigma}[p_{\sigma\tau}Finish_\tau] = Farm_\varrho \wedge \\
&LocationOfOstrich_{\varrho\sigma}[p_{\sigma\tau}Finish_\tau] = Farm_\varrho \wedge \\
&LocationOfCheetah_{\varrho\sigma}[p_{\sigma\tau}Finish_\tau] = Farm_\varrho \wedge \\
&\forall t_\tau\ [Moments_{o\tau}t_\tau \supset \\
&\quad [[LocationOfOstrich_{\varrho\sigma}[p_{\sigma\tau}t_\tau] = LocationOfCheetah_{\varrho\sigma}[p_{\sigma\tau}t_\tau]] \supset \\
&\quad [LocationOfSheila_{\varrho\sigma}[p_{\sigma\tau}t_\tau] = LocationOfCheetah_{\varrho\sigma}[p_{\sigma\tau}t_\tau]]]]]
\end{aligned}$$

We can express the assertion that Sheila has a way to accomplish her objective with the formula

$$\exists p_{\sigma\tau}[Proposals_{o(\sigma\tau)}p_{\sigma\tau} \wedge Acceptable_{o(\sigma\tau)}p_{\sigma\tau}]$$

Example 3 We now provide a mathematical example. Mathematical ideas can be expressed in type theory without introducing any new constants. An *iterate* of a function f from a set to itself is a function which applies f one or more times. For example, if $g(x) = f(f(f(x)))$, then g is an iterate of f. $[\text{ITERATE}+_{o(\iota)(\iota)}f_\iota g_\iota]$ means that g_ι is an iterate of f_ι. $\text{ITERATE}+_{o(\iota)(\iota)}$

is defined (inductively) as

$$\lambda f_{\iota\iota}\lambda g_{\iota\iota}\forall p_{o(\iota\iota)}[p_{o(\iota\iota)}f_{\iota\iota} \wedge \forall j_{\iota\iota}[p_{o(\iota\iota)}j_{\iota\iota} \supset p_{o(\iota\iota)}[\lambda x_\iota f_{\iota\iota}[j_{\iota\iota}x_\iota]]] \supset p_{o(\iota\iota)}g_{\iota\iota}]$$

Thus, g is an iterate of f if g is in every set p of functions which contains f and which contains the function $\lambda x_\iota f_{\iota\iota}[j_{\iota\iota}x_\iota]$ (i.e., f composed with j) whenever it contains j.

A *fixed point* of f is an element y such that $f(y) = y$.

It can be proved that if some iterate of a function f has a unique fixed point, then f itself has a fixed point. This theorem can be expressed by the wff

$$\begin{aligned}&\forall f_{\iota\iota}[\exists g_{\iota\iota}[\mathrm{ITERATE}{+}_{o(\iota\iota)(\iota\iota)}f_{\iota\iota}g_{\iota\iota} \wedge \\ &\quad \exists x_\iota[g_{\iota\iota}x_\iota = x_\iota \wedge \forall z_\iota[g_{\iota\iota}z_\iota = z_\iota \supset z_\iota = x_\iota]]] \supset \\ &\quad \exists y_\iota[f_{\iota\iota}y_\iota = y_\iota]].\end{aligned}$$

See Andrews *et al.* 1996, for a discussion of how this theorem, which is called THM15B, can be proved automatically.

Example 4 An example from philosophy is Gödel's variant of the ontological argument for the existence of God. This example illustrates two interesting aspects:

- Church's type theory can be employed as a meta-logic to concisely embed expressive other logics such as the higher-order modal logic assumed by Gödel. By exploiting the possible world semantics of this target logic, its syntactic elements are defined in such a way that the infrastructure of the meta-logic are reused as much as possible. In this technique, called *shallow semantical embedding*, the modal operator $\Box$, for example, is simply identified with (taken as syntactic sugar for) the λ-formula

  $$\lambda\varphi_{o\iota}\lambda w_\iota\forall v_\iota[R_{o\iota\iota}w_\iota v_\iota \supset \varphi_{o\iota}v_\iota]$$

  where $R_{o\iota\iota}$ denotes the accessibility relation associated with $\Box$ and type ι is identified with possible worlds. Moreover, since $\forall x_\alpha[\mathbf{A}_{o\alpha}x_\alpha]$ is shorthand in Church's type theory for $\Pi_{o(o\alpha)}[\lambda x_\alpha[\mathbf{A}_{o\alpha}x_\alpha]]$, the modal formula

  $$\Box\forall x\mathbf{P}x$$

  is represented as

  $$\Box\Pi'[\lambda x_\alpha\lambda w_\iota[\mathbf{P}_{o\iota\alpha}x_\alpha w_\iota]]$$

  where Π' stands for the λ-term

  $$\lambda\Phi_{o\iota\alpha}\lambda w_\iota\forall x_\alpha[\Phi_{o\iota\alpha}x_\alpha w_\iota]$$

  and the $\Box$ gets resolved as described above. The above choice of Π' realizes a possibilist notion of quantification. By introducing a binary 'existence' predicate in the meta-logic and by utilizing this predicate as an additional

guard in the definition of Π' an actualist notion of quantification can be obtained. Expressing that an embedded modal formula φ_{oi} is globally valid is then captured by the formula $\forall x_i[\varphi_{oi}x_i]$. Local validity (and also actuality) can be modeled as $\varphi_{oi}n_i$, where n_i is a nominal (constant symbol in the meta-logic) denoting a particular possible world.

- The above technique can be exploited for a natural encoding and automated assessment of Gödel's ontological argument in higher-order modal logic, which unfolds into formulas in Church's type theory such that higher-order theorem provers can be applied. Further details are presented in Section 6 (Logic and Philosophy) of the SEP entry on automated reasoning and also in Section 5.3; moreover, see Benzmüller & Woltzenlogel-Paleo 2014 and Benzmüller 2019.

Example 5 Suppose we omit the use of type symbols in the definitions of wffs. Then we can write the formula $\lambda x \sim [xx]$, which we shall call R. It can be regarded as denoting the set of all sets x such that x is not in x. We may then consider the formula $[R\ R]$, which expresses the assertion that R is in itself. We can clearly prove $[R\ R] \equiv [[\lambda x \sim [xx]]R]$, so by λ-conversion we can derive $[R\ R] \equiv \sim [R\ R]$, which is a contradiction. This is Russell's paradox. Russell's discovery of this paradox (Russell 1903, 101-107) played a crucial role in the development of type theory. Of course, when type symbols are present, R is not well-formed, and the contradiction cannot be derived.

1.3 Axioms and Rules of Inference

1.3.1 Rules of Inference

1. *Alphabetic Change of Bound Variables* (α-*conversion*). To replace any well-formed part $\lambda\mathbf{x}_\beta\mathbf{A}_\alpha$ of a wff by $\lambda\mathbf{y}_\beta\mathsf{Sub}(\mathbf{y}_\beta, \mathbf{x}_\beta, \mathbf{A}_\alpha)$, provided that $\mathbf{y}_\beta$ does not occur in $\mathbf{A}_\alpha$ and $\mathbf{x}_\beta$ is not bound in $\mathbf{A}_\alpha$.

2. β-*contraction*. To replace any well-formed part $[\lambda\mathbf{x}_\alpha\mathbf{B}_\beta]\mathbf{A}_\alpha$ of a wff by $\mathsf{Sub}(\mathbf{A}_\alpha, \mathbf{x}_\alpha, \mathbf{B}_\beta)$, provided that the bound variables of $\mathbf{B}_\beta$ are distinct both from $\mathbf{x}_\alpha$ and from the free variables of $\mathbf{A}_\alpha$.

3. β-*expansion*. To infer $\mathbf{C}$ from $\mathbf{D}$ if $\mathbf{D}$ can be inferred from $\mathbf{C}$ by a single application of β-contraction.

4. *Substitution*. From $\mathbf{F}_{(o\alpha)}\mathbf{x}_\alpha$, to infer $\mathbf{F}_{(o\alpha)}\mathbf{A}_\alpha$, provided that $\mathbf{x}_\alpha$ is not a free variable of $\mathbf{F}_{(o\alpha)}$.

5. *Modus Ponens*. From $[\mathbf{A}_o \supset \mathbf{B}_o]$ and $\mathbf{A}_o$, to infer $\mathbf{B}_o$.

6. *Generalization*. From $\mathbf{F}_{(o\alpha)}\mathbf{x}_\alpha$ to infer $\Pi_{o(o\alpha)}\mathbf{F}_{(o\alpha)}$, provided that $\mathbf{x}_\alpha$ is not a free variable of $\mathbf{F}_{(o\alpha)}$.

1.3.2 Elementary Type Theory

We start by listing the axioms for what we shall call *elementary type theory.*

$$
\begin{array}{ll}
(1) & [p_o \vee p_o] \supset p_o \\
(2) & p_o \supset [p_o \vee q_o] \\
(3) & [p_o \vee q_o] \supset [q_o \vee p_o] \\
(4) & [p_o \supset q_o] \supset [[r_o \vee p_o] \supset [r_o \vee q_o]] \\
(5^\alpha) & \Pi_{o(o\alpha)} f_{(o\alpha)} \supset f_{(o\alpha)} x_\alpha \\
(6^\alpha) & \forall x_\alpha [p_o \vee f_{(o\alpha)} x_\alpha] \supset [p_o \vee \Pi_{o(o\alpha)} f_{(o\alpha)}]
\end{array}
$$

The theorems of elementary type theory are those theorems which can be derived, using the rules of inference, from Axioms (1)–(6^α) (for all type symbols α). We shall sometimes refer to elementary type theory as $\mathcal{T}$. It embodies the logic of propositional connectives, quantifiers, and λ-conversion in the context of type theory.

To illustrate the rules and axioms introduced above, we give a short and trivial proof in $\mathcal{T}$. Following each wff of the proof, we indicate how it was inferred. (The proof is actually quite inefficient, since line 3 is not used later, and line 7 can be derived directly from line 5 without using line 6. The additional proof lines have been inserted to illustrate some relevant aspects. For the sake of readability, many brackets have been deleted from the formulas in this proof. The diligent reader should be able to restore them.)

1.	$\forall x_\iota [p_o \vee f_{o\iota} x_\iota] \supset [p_o \vee \Pi_{o(o\iota)} f_{o\iota}]$	Axiom 6^ι
2.	$[\lambda f_{o\iota} [\forall x_\iota [p_o \vee f_{o\iota} x_\iota] \supset [p_o \vee \Pi_{o(o\iota)} f_{o\iota}]]] f_{o\iota}$	β-expansion: 1
3.	$\Pi_{o(o(o\iota))} [\lambda f_{o\iota} [\forall x_\iota [p_o \vee f_{o\iota} x_\iota] \supset [p_o \vee \Pi_{o(o\iota)} f_{o\iota}]]]$	Generalization: 2
4.	$[\lambda f_{o\iota} [\forall x_\iota [p_o \vee f_{o\iota} x_\iota] \supset [p_o \vee \Pi_{o(o\iota)} f_{o\iota}]]] [\lambda x_\iota r_{o\iota} x_\iota]$	Substitution: 2
5.	$\forall x_\iota [p_o \vee [\lambda x_\iota r_{o\iota} x_\iota] x_\iota] \supset [p_o \vee \Pi_{o(o\iota)} [\lambda x_\iota r_{o\iota} x_\iota]]$	β-contraction: 4
6.	$\forall x_\iota [p_o \vee [\lambda y_\iota r_{o\iota} y_\iota] x_\iota] \supset [p_o \vee \Pi_{o(o\iota)} [\lambda x_\iota r_{o\iota} x_\iota]]$	α-conversion: 5
7.	$\forall x_\iota [p_o \vee r_{o\iota} x_\iota] \supset [p_o \vee \Pi_{o(o\iota)} [\lambda x_\iota r_{o\iota} x_\iota]]$	β-contraction: 6

Note that (3) can be written as

$$(3') \qquad \forall f_{o\iota} [\forall x_\iota [p_o \vee f_{o\iota} x_\iota] \supset [p_o \vee [\forall x_\iota f_{o\iota} x_\iota]]]$$

and (7) can be written as

$$(7') \qquad \forall x_\iota [p_o \vee r_{o\iota} x_\iota] \supset [p_o \vee \forall x_\iota r_{o\iota} x_\iota]$$

We have thus derived a well known law of quantification theory. We illustrate one possible interpretation of the wff ($7'$) (which is closely related to Axiom 6) by considering a situation in which a rancher puts some horses in a corral and leaves for the night. Later, he cannot remember whether he closed the gate to the corral. While reflecting on the situation, he comes to a conclusion which can

be expressed by $(7')$ if we take the horses to be the elements of type $\imath$, interpret p_o to mean "the gate was closed", and interpret $r_{o\imath}$ so that $r_{o\imath}x_\imath$ asserts "$x_\imath$ left the corral". With this interpretation, $(7')$ says "If it is true of every horse that the gate was closed or that the horse left the corral, then the gate was closed or every horse left the corral."

To the axioms listed above we add the axioms below to obtain Church's type theory.

1.3.3 Axioms of Extensionality

The axioms of boolean and functional extensionality are the following:

$$(7^o) \qquad [x_o \equiv y_o] \supset x_o = y_o$$
$$(7^{\alpha\beta}) \qquad \forall x_\beta [f_{\alpha\beta}x_\beta = g_{\alpha\beta}x_\beta] \supset f_{\alpha\beta} = g_{\alpha\beta}$$

Church did not include Axiom 7^o in his list of axioms in Church 1940, but he mentioned the possibility of including it. Henkin did include it in Henkin 1950.

1.3.4 Descriptions

The expression

$$\exists_1 \mathbf{x}_\alpha \mathbf{A}_o$$

stands for

$$[\lambda p_{o\alpha} \exists y_\alpha [p_{o\alpha}y_\alpha \wedge \forall z_\alpha [p_{o\alpha}z_\alpha \supset z_\alpha = y_\alpha]]] \, [\lambda \mathbf{x}_\alpha \mathbf{A}_o]$$

For example,

$$\exists_1 x_\alpha P_{o\alpha}x_\alpha$$

stands for

$$[\lambda p_{o\alpha} \exists y_\alpha [p_{o\alpha}y_\alpha \wedge \forall z_\alpha [p_{o\alpha}z_\alpha \supset z_\alpha = y_\alpha]]] \, [\lambda x_\alpha P_{o\alpha}x_\alpha]$$

By λ-conversion, this is equivalent to

$$\exists y_\alpha [[\lambda x_\alpha P_{o\alpha}x_\alpha]y_\alpha \wedge \forall z_\alpha [[\lambda x_\alpha P_{o\alpha}x_\alpha]z_\alpha \supset z_\alpha = y_\alpha]]$$

which reduces by λ-conversion to

$$\exists y_\alpha [P_{o\alpha}y_\alpha \wedge \forall z_\alpha [P_{o\alpha}z_\alpha \supset z_\alpha = y_\alpha]]$$

This asserts that there is a unique element which has the property $P_{o\alpha}$. From this example we can see that in general, $\exists_1 \mathbf{x}_\alpha \mathbf{A}_o$ expresses the assertion that "there is a unique $\mathbf{x}_\alpha$ such that $\mathbf{A}_o$".

When there is a unique such element $\mathbf{x}_\alpha$, it is convenient to have the notation $\imath \mathbf{x}_\alpha \mathbf{A}_o$ to represent the expression "the $\mathbf{x}_\alpha$ such that $\mathbf{A}_o$". Russell showed in Whitehead & Russell 1927b how to provide contextual definitions for such notations in his formulation of type theory. In Church's type theory $\imath \mathbf{x}_\alpha \mathbf{A}_o$ is

defined as $\iota_{\alpha(o\alpha)}[\lambda \mathbf{x}_\alpha \mathbf{A}_o]$. Thus, $\imath$ behaves like a variable-binding operator, but it is defined in terms of λ with the aid of the constant $\iota_{\alpha(o\alpha)}$. Thus, λ is still the only variable-binding operator that is needed.

Since $\mathbf{A}_o$ describes $\mathbf{x}_\alpha, \iota_{\alpha(o\alpha)}$ is called a *description operator*. Associated with this notation is the following:

Axiom of Descriptions:

$$(8^\alpha) \qquad \exists_1 x_\alpha [p_{o\alpha} x_\alpha] \supset p_{o\alpha}[\iota_{\alpha(o\alpha)} p_{o\alpha}]$$

This says that when the set $p_{o\alpha}$ has a unique member, then $\iota_{\alpha(o\alpha)} p_{o\alpha}$ is in $p_{o\alpha}$, and therefore is that unique member. Thus, this axiom asserts that $\iota_{\alpha(o\alpha)}$ maps one-element sets to their unique members.

If from certain hypotheses one can prove

$$\exists_1 \mathbf{x}_\alpha \mathbf{A}_o$$

then by using Axiom 8^α one can derive

$$[\lambda \mathbf{x}_\alpha \mathbf{A}_o][\iota_{\alpha(o\alpha)}[\lambda \mathbf{x}_\alpha \mathbf{A}_o]]$$

which can also be written as

$$[\lambda \mathbf{x}_\alpha \mathbf{A}_o][\imath \mathbf{x}_\alpha \mathbf{A}_o]$$

We illustrate the usefulness of the description operator with a small example. Suppose we have formalized the theory of real numbers, and our theory has constants 1_ϱ and $\times_{\varrho\varrho\varrho}$ to represent the number 1 and the multiplication function, respectively. (Here ϱ is the type of real numbers.) To represent the multiplicative inverse function, we can define the wff $INV_{\varrho\varrho}$ as

$$\lambda z_\varrho \imath x_\varrho [\times_{\varrho\varrho\varrho} z_\varrho x_\varrho = 1_\varrho]$$

Of course, in traditional mathematical notation we would not write the type symbols, and we would write $\times_{\varrho\varrho\varrho} z_\varrho x_\varrho$ as $z \times x$ and write $INV_{\varrho\varrho} z$ as z^{-1}. Thus z^{-1} is defined to be that x such that $z \times x = 1$. When Z is provably not 0, we will be able to prove $\exists_1 x_\varrho [\times_{\varrho\varrho\varrho} Z\, x_\varrho = 1_\varrho]$ and $Z \times Z^{-1} = 1$, but if we cannot establish that Z is not 0, nothing significant about Z^{-1} will be provable.

1.3.5 Axiom of Choice

The Axiom of Choice can be expressed as follows in Church's type theory:

$$(9^\alpha) \qquad \exists x_\alpha p_{o\alpha} x_\alpha \supset p_{o\alpha}[\iota_{\alpha(o\alpha)} p_{o\alpha}]$$

(9^α) says that the choice function $\iota_{\alpha(o\alpha)}$ chooses from every nonempty set $p_{o\alpha}$ an element, designated as $\iota_{\alpha(o\alpha)} p_{o\alpha}$, of that set. When this form of the Axiom of Choice is included in the list of axioms, $\iota_{\alpha(o\alpha)}$ is called a selection operator instead of a description operator, and $\imath \mathbf{x}_\alpha \mathbf{A}_o$ means "an $\mathbf{x}_\alpha$ such that

$\mathbf{A}_o$" when there is some such element $\mathbf{x}_\alpha$. These selection operators have the same meaning as Hilbert's ϵ-operator (Hilbert 1928). However, we here provide one such operator for each type α.

It is natural to call $\imath$ a definite description operator in contexts where $\imath\mathbf{x}_\alpha\mathbf{A}_o$ means "the $\mathbf{x}_\alpha$ such that $\mathbf{A}_o$", and to call it an indefinite description operator in contexts where $\imath\mathbf{x}_\alpha\mathbf{A}_o$ means "an $\mathbf{x}_\alpha$ such that $\mathbf{A}_o$".

Clearly the Axiom of Choice implies the Axiom of Descriptions, but sometimes formulations of type theory are used which include the Axiom of Descriptions, but not the Axiom of Choice.

Another formulation of the Axiom of Choice simply asserts the existence of a choice function without explicitly naming it:

$$(\mathrm{AC}^\alpha) \qquad \exists j_{\alpha(o\alpha)}\forall p_{o\alpha}[\exists x_\alpha p_{o\alpha}x_\alpha \supset p_{o\alpha}[j_{\alpha(o\alpha)}p_{o\alpha}]]$$

Normally when one assumes the Axiom of Choice in type theory, one assumes it as an axiom schema, and asserts AC^α for each type symbol α. A similar remark applies to the axioms for extensionality and description. However, modern proof systems for Church's type theory, which are e.g. based on resolution, do in fact avoid the addition of such axiom schemata for reasons as further explained in Sections 3.4 and 4 below. They work with more constrained, goal-directed proof rules instead.

Before proceeding, we need to introduce some terminology. $\mathcal{Q}_0$ is an alternative formulation of Church's type theory which will be described in Section 1.4 and is equivalent to the system described above using Axioms (1)–(8). A type symbol is *propositional* if the only symbols which occur in it are o and parentheses.

Yasuhara (1975) defined the relation '$\geq$' between types as the reflexive transitive closure of the minimal relation such that $(\alpha\beta) \geq \alpha$ and $(\alpha\beta) \geq \beta$. He established that:

- If $\alpha \geq \beta$, then $\mathcal{Q}_0 \vdash \mathrm{AC}^\alpha \supset \mathrm{AC}^\beta$.
- Given a set S of types, none of which is propositional, there is a model of $\mathcal{Q}_0$ in which AC^α fails if and only if $\alpha \geq \beta$ for some β in S.

The existence of a choice functions for "higher" types thus entails the existence of choice functions for "lower" types, the opposite is generally not the case though.

Büchi (1953) has shown that while the schemas expressing the Axiom of Choice and Zorn's Lemma can be derived from each other, the relationships between the particular types involved are complex.

1.3.6 Axioms of Infinity

One can define the natural numbers (and therefore other basic mathematical structures such as the real and complex numbers) in type theory, but to prove that they have the required properties (such as Peano's Postulates), one needs

an Axiom of Infinity. There are many viable possibilities for such an axiom, such as those discussed in Church 1940, section 57 of Church 1956, and section 60 of Andrews 2002.

1.4 A Formulation Based on Equality

In Section 1.2.1, $\sim_{(oo)}, \vee_{((oo)o)}$, and the $\Pi_{(o(o\alpha))}$'s were taken as primitive constants, and the wffs $\mathsf{Q}_{o\alpha\alpha}$ which denote equality relations at type α were defined in terms of these. We now present an alternative formulation $\mathcal{Q}_0$ of Church's type theory in which there are primitive constants $\mathsf{Q}_{o\alpha\alpha}$ denoting equality, and $\sim_{(oo)}, \vee_{((oo)o)}$, and the $\Pi_{(o(o\alpha))}$'s are defined in terms of the $\mathsf{Q}_{o\alpha\alpha}$'s.

Tarski (1923) noted that in the context of higher-order logic, one can define propositional connectives in terms of logical equivalence and quantifiers. Quine (1956) showed how both quantifiers and connectives can be defined in terms of equality and the abstraction operator λ in the context of Church's type theory. Henkin (1963) rediscovered these definitions, and developed a formulation of Church's type theory based on equality in which he restricted attention to propositional types. Andrews (1963) simplified the axioms for this system.

$\mathcal{Q}_0$ is based on these ideas, and can be shown to be equivalent to a formulation of Church's type theory using Axioms (1)–(8) of the preceding sections. This section provides an alternative to the material in the preceding Sections 1.2.1–1.3.4. More details about $\mathcal{Q}_0$ can be found in Andrews 2002.

1.4.1 Definitions

- Type symbols, improper symbols, and variables of $\mathcal{Q}_0$ are defined as in Section 1.2.1.
- The logical constants of $\mathcal{Q}_0$ are $\mathsf{Q}_{((o\alpha)\alpha)}$ and $\iota_{(\iota(o\iota))}$ (for each type symbol α).
- Wffs of $\mathcal{Q}_0$ are defined as in Section 1.2.1.

Abbreviations:

- $\mathbf{A}_\alpha = \mathbf{B}_\alpha$ stands for $\mathsf{Q}_{o\alpha\alpha}\mathbf{A}_\alpha\mathbf{B}_\alpha$
- $\mathbf{A}_o \equiv \mathbf{B}_o$ stands for $\mathsf{Q}_{ooo}\mathbf{A}_o\mathbf{B}_o$
- T_o stands for $\mathsf{Q}_{ooo} = \mathsf{Q}_{ooo}$
- F_o stands for $[\lambda x_o T_o] = [\lambda x_o x_o]$
- $\Pi_{o(o\alpha)}$ stands for $\mathsf{Q}_{o(o\alpha)(o\alpha)}[\lambda x_\alpha T_o]$
- $\forall \mathbf{x}_\alpha \mathbf{A}$ stands for $\Pi_{o(o\alpha)}[\lambda \mathbf{x}_\alpha \mathbf{A}]$
- $\wedge_{ooo}$ stands for $\lambda x_o \lambda y_o[[\lambda g_{ooo}[g_{ooo}T_oT_o]] = [\lambda g_{ooo}[g_{ooo}x_oy_o]]]$
- $\mathbf{A}_o \wedge \mathbf{B}_o$ stands for $\wedge_{ooo}\mathbf{A}_o\mathbf{B}_o$

- $\sim_{oo}$ stands for $\mathsf{Q}_{ooo}F_o$

T_o denotes truth. The meaning of $\Pi_{o(o\alpha)}$ was discussed in Section 1.1. To see that this definition of $\Pi_{o(o\alpha)}$ is appropriate, note that $\lambda x_\alpha T$ denotes the set of all elements of type α, and that $\Pi_{o(o\alpha)}s_{o\alpha}$ stands for $\mathsf{Q}_{o(o\alpha)(o\alpha)}[\lambda x_\alpha T]s_{o\alpha}$, respectively for $[\lambda x_\alpha T] = s_{o\alpha}$. Therefore $\Pi_{o(o\alpha)}s_{o\alpha}$ asserts that $s_{o\alpha}$ is the set of all elements of type α, so $s_{o\alpha}$ contains all elements of type α. It can be seen that F_o can also be written as $\forall x_o x_o$, which asserts that everything is true. This is false, so F_o denotes falsehood. The expression $\lambda g_{ooo}[g_{ooo}x_o y_o]$ can be used to represent the ordered pair $\langle x_o, y_o\rangle$, and the conjunction $x_o \wedge y_o$ is true iff x_o and y_o are both true, i.e., iff $\langle T_o, T_o\rangle = \langle x_o, y_o\rangle$. Hence $x_o \wedge y_o$ can be expressed by the formula $[\lambda g_{ooo}[g_{ooo}T_o T_o]] = [\lambda g_{ooo}[g_{ooo}x_o y_o]]$.

Other propositional connectives and the existential quantifier are easily defined. By using $\iota_{(\iota(o\iota))}$, one can define description operators $\iota_{\alpha(o\alpha)}$ for all types α.

1.4.2 Axioms and Rules of Inference

$\mathcal{Q}_0$ has a single rule of inference.

Rule R: From $\mathbf{C}$ and $\mathbf{A}_\alpha = \mathbf{B}_\alpha$, to infer the result of replacing one occurrence of $\mathbf{A}_\alpha$ in $\mathbf{C}$ by an occurrence of $\mathbf{B}_\alpha$, provided that the occurrence of $\mathbf{A}_\alpha$ in $\mathbf{C}$ is not (an occurrence of a variable) immediately preceded by λ.

The *axioms for* $\mathcal{Q}_0$ are the following:

(1) $[g_{oo}T_o \wedge g_{oo}F_o] = \forall x_o[g_{oo}x_o]$

(2^α) $[x_\alpha = y_\alpha] \supset [h_{o\alpha}x_\alpha = h_{o\alpha}y_\alpha]$

($3^{\alpha\beta}$) $[f_{\alpha\beta} = g_{\alpha\beta}] = \forall x_\beta[f_{\alpha\beta}x_\beta = g_{\alpha\beta}x_\beta]$

(4) $[\lambda \mathbf{x}_\alpha \mathbf{B}_\beta]\mathbf{A}_\alpha = \mathsf{SubFree}(\mathbf{A}_\alpha, \mathbf{x}_\alpha, \mathbf{B}_\beta)$, provided $\mathbf{A}_\alpha$ is free for $\mathbf{x}$ in $\mathbf{B}_\beta$.

(5) $\iota_{\iota(o\iota)}[\mathsf{Q}_{o\iota\iota}y_\iota] = y_\iota$

2 Semantics

It is natural to compare the semantics of type theory with the semantics of first-order logic, where the theorems are precisely the wffs which are valid in all interpretations. From an intuitive point of view, the natural interpretations of type theory are *standard models*, which are defined below. However, it is a consequence of Gödel's Incompleteness Theorem (Gödel 1931) that axioms (1)–(9) do not suffice to derive all wffs which are valid in all standard models, and there is no consistent recursively axiomatized extension of these axioms which suffices for this purpose. Nevertheless, experience shows that these axioms are sufficient for most purposes, and Leon Henkin considered the problem of clarifying in what sense they are complete. The definitions and theorem below constitute Henkin's (1950) solution to this problem, which is often referred to as *general semantics* or *Henkin semantics*.

A *frame* is a collection $\{\mathcal{D}_\alpha\}_\alpha$ of nonempty domains (sets) $\mathcal{D}_\alpha$, one for each type symbol α, such that $\mathcal{D}_o = \{\mathsf{T}, \mathsf{F}\}$ (where T represents truth and F represents falsehood), and $\mathcal{D}_{\alpha\beta}$ is some collection of functions mapping $\mathcal{D}_\beta$ into $\mathcal{D}_\alpha$. The members of $\mathcal{D}_\iota$ are called *individuals*.

An *interpretation* $\langle\{\mathcal{D}_\alpha\}_\alpha, \mathfrak{I}\rangle$ consists of a frame and a function $\mathfrak{I}$ which maps each constant C of type α to an appropriate element of $\mathcal{D}_\alpha$, which is called the *denotation of* C. The logical constants are given their standard denotations.

An *assignment* of values in the frame $\{\mathcal{D}_\alpha\}_\alpha$ to variables is a function ϕ such that $\phi\mathbf{x}_\alpha \in \mathcal{D}_\alpha$ for each variable $\mathbf{x}_\alpha$. (Notation: The assignment $\phi[a/x]$ maps variable x to value a and it is identical with ϕ for all other variable symbols different from x.)

An interpretation $\mathcal{M} = \langle\{\mathcal{D}_\alpha\}_\alpha, \mathfrak{I}\rangle$ is a *general model* (aka *Henkin model*) iff there is a binary function $\mathcal{V}$ such that $\mathcal{V}_\phi\mathbf{A}_\alpha \in \mathcal{D}_\alpha$ for each assignment ϕ and wff $\mathbf{A}_\alpha$, and the following conditions are satisfied for all assignments and all wffs:

- $\mathcal{V}_\phi\mathbf{x}_\alpha = \phi\mathbf{x}_\alpha$ for each variable $\mathbf{x}_\alpha$.
- $\mathcal{V}_\phi A_\alpha = \mathfrak{I}A_\alpha$ if A_α is a primitive constant.
- $\mathcal{V}_\phi[\mathbf{A}_{\alpha\beta}\mathbf{B}_\beta] = (\mathcal{V}_\phi\mathbf{A}_{\alpha\beta})(\mathcal{V}_\phi\mathbf{B}_\beta)$ (the value of a function $\mathcal{V}_\phi\mathbf{A}_{\alpha\beta}$ at the argument $\mathcal{V}_\phi\mathbf{B}_\beta$).
- $\mathcal{V}_\phi[\lambda\mathbf{x}_\alpha\mathbf{B}_\beta]$ = that function from $\mathcal{D}_\alpha$ into $\mathcal{D}_\beta$ whose value for each argument $z \in \mathcal{D}_\alpha$ is $\mathcal{V}_\psi\mathbf{B}_\beta$, where ψ is that assignment such that $\psi\mathbf{x}_\alpha = z$ and $\psi\mathbf{y}_\beta = \phi\mathbf{y}_\beta$ if $\mathbf{y}_\beta \neq \mathbf{x}_\alpha$.

If an interpretation $\mathcal{M}$ is a general model, the function $\mathcal{V}$ is uniquely determined. $\mathcal{V}_\phi\mathbf{A}_\alpha$ is called the *value* of $\mathbf{A}_\alpha$ in $\mathcal{M}$ with respect to ϕ.

One can easily show that the following statements hold in all general models $\mathcal{M}$ for all assignments ϕ and all wffs $\mathbf{A}$ and $\mathbf{B}$:

- $\mathcal{V}_\phi T_o = \mathsf{T}$ and $\mathcal{V}_\phi F_o = \mathsf{F}$
- $\mathcal{V}_\phi[\sim_{oo} \mathbf{A}_o] = \mathsf{T}$ iff $\mathcal{V}_\phi\mathbf{A}_o = \mathsf{F}$
- $\mathcal{V}_\phi[\mathbf{A}_o \vee \mathbf{B}_o] = \mathsf{T}$ iff $\mathcal{V}_\phi\mathbf{A}_o = \mathsf{T}$ or $\mathcal{V}_\phi\mathbf{B}_o = \mathsf{T}$
- $\mathcal{V}_\phi[\mathbf{A}_o \wedge \mathbf{B}_o] = \mathsf{T}$ iff $\mathcal{V}_\phi\mathbf{A}_o = \mathsf{T}$ and $\mathcal{V}_\phi\mathbf{B}_o = \mathsf{T}$
- $\mathcal{V}_\phi[\mathbf{A}_o \supset \mathbf{B}_o] = \mathsf{T}$ iff $\mathcal{V}_\phi\mathbf{A}_o = \mathsf{F}$ or $\mathcal{V}_\phi\mathbf{B}_o = \mathsf{T}$
- $\mathcal{V}_\phi[\mathbf{A}_o \equiv \mathbf{B}_o] = \mathsf{T}$ iff $\mathcal{V}_\phi\mathbf{A}_o = \mathcal{V}_\phi\mathbf{B}_o$
- $\mathcal{V}_\phi[\forall\mathbf{x}_\alpha\mathbf{A}] = \mathsf{T}$ iff $\mathcal{V}_{\phi[a/x]}\mathbf{A} = \mathsf{T}$ for all $a \in \mathcal{D}_\alpha$
- $\mathcal{V}_\phi[\exists\mathbf{x}_\alpha\mathbf{A}] = \mathsf{T}$ iff there exists an $a \in \mathcal{D}_\alpha$ such that $\mathcal{V}_{\phi[a/x]}\mathbf{A} = \mathsf{T}$

The semantics of general models is thus as expected. However, there is a subtlety to note regarding the following condition for arbitrary types α:

- $\mathcal{V}_\phi[\mathbf{A}_\alpha = \mathbf{B}_\alpha] = \mathsf{T}$ iff $\mathcal{V}_\phi\mathbf{A}_\alpha = \mathcal{V}_\phi\mathbf{B}_\alpha$

When the definitions of Section 1.2.1 are employed, where equality has been defined in terms of Leibniz' principle, then this statement is not implied for all types α. It only holds if we additionally require that the domains $\mathcal{D}_{o\alpha}$ contain all the unit sets of objects of type α, or, alternatively, that the domains $\mathcal{D}_{o\alpha\alpha}$ contain the respective identity relations on objects of type α (which entails the former). The need for this additional requirement, which is not included in the original work of Henkin (1950), has been demonstrated in Andrews 1972a.

When instead the alternative definitions of Section 1.4 are employed, then this requirement is obviously met due to the presence of the logical constants $\mathsf{Q}_{o\alpha\alpha}$ in the signature, which by definition denote the respective identity relations on the objects of type α and therefore trivially ensure their existence in each general model $\mathcal{M}$. It is therefore a natural option to always assume primitive equality constants (for each type α) in a concrete choice of base system for Church's type theory, just as realised in Andrews' system $\mathcal{Q}_0$.

An interpretation $\langle\{\mathcal{D}_\alpha\}_\alpha, \mathfrak{I}\rangle$ is a *standard model* iff for all α and $\beta, \mathcal{D}_{\alpha\beta}$ is the set of all functions from $\mathcal{D}_\beta$ into $\mathcal{D}_\alpha$. Clearly a standard model is a general model.

We say that a wff $\mathbf{A}$ is *valid* in a model $\mathcal{M}$ iff $\mathcal{V}_\phi\mathbf{A} = \mathsf{T}$ for every assignment ϕ into $\mathcal{M}$. A model for a set $\mathcal{H}$ of wffs is a model in which each wff of $\mathcal{H}$ is valid.

A wff $\mathbf{A}$ is *valid in the general* [*standard*] *sense* iff $\mathbf{A}$ is valid in every general [standard] model. Clearly a wff which is valid in the general sense is valid in the standard sense, but the converse of this statement is false.

Completeness and Soundness Theorem (Henkin 1950):
A wff is a theorem if and only if it is valid in the general sense.

Not all frames belong to interpretations, and not all interpretations are general models. In order to be a general model, an interpretation must have a frame satisfying certain closure conditions which are discussed further in Andrews 1972b. Basically, in a general model every wff must have a value with respect to each assignment.

A model is said to be *finite* iff its domain of individuals is finite. Every finite model for $\mathcal{Q}_0$ is standard (Andrews 2002, Theorem 5404), but every set of sentences of $\mathcal{Q}_0$ which has infinite models also has nonstandard models (Andrews2002, Theorem 5506).

An understanding of the distinction between standard and nonstandard models can clarify many phenomena. For example, it can be shown that there is a model $\mathcal{M} = \langle\{\mathcal{D}_\alpha\}_\alpha, \mathfrak{I}\rangle$ in which $\mathcal{D}_\iota$ is infinite, and all the domains $\mathcal{D}_\alpha$ are countable. Thus $\mathcal{D}_\iota$ and $\mathcal{D}_{o\iota}$ are both countably infinite, so there must be a bijection h between them. However, Cantor's Theorem (which is provable in type theory and therefore valid in all models) says that $\mathcal{D}_\iota$ has more subsets than members. This seemingly paradoxical situation is called Skolem's Paradox. It can be resolved by looking carefully at Cantor's Theorem, i.e., $\sim \exists g_{o\iota\iota}\forall f_{o\iota}\exists j_\iota[g_{o\iota\iota}j_\iota = f_{o\iota}]$, and considering what it means in a model. The

theorem says that there is no function $g \in \mathcal{D}_{o\iota\iota}$ from $\mathcal{D}_\iota$ into $\mathcal{D}_{o\iota}$ which has every set $f_{o\iota} \in \mathcal{D}_{o\iota}$ in its range. The usual interpretation of the statement is that $\mathcal{D}_{o\iota}$ is bigger (in cardinality) than $\mathcal{D}_\iota$. However, what it actually means in this model is that h cannot be in $\mathcal{D}_{o\iota\iota}$. Of course, $\mathcal{M}$ must be nonstandard.

While the Axiom of Choice is presumably true in all standard models, there is a nonstandard model for $\mathcal{Q}_0$ in which AC^ι is false (Andrews 1972b). Thus, AC^ι is not provable in $\mathcal{Q}_0$.

Thus far, investigations of model theory for Church's type theory have been far less extensive than for first-order logic. Nevertheless, there has been some work on methods of constructing nonstandard models of type theory and models in which various forms of extensionality fail, models for theories with arbitrary (possibly incomplete) sets of logical constants, and on developing general methods of establishing completeness of various systems of axioms with respect to various classes of models. Relevant papers include Andrews 1971, 1972a,b, and Henkin 1975. Further related work can be found in Benzmüller *et al.* 2004, Brown 2004, 2007, and Muskens 2007.

3 Metatheory

3.1 Lambda-Conversion

The first three rules of inference in Section 1.3.1 are called rules of λ-*conversion*. If $\mathbf{D}$ and $\mathbf{E}$ are wffs, we write $\mathbf{D}$ conv $\mathbf{E}$ to indicate that $\mathbf{D}$ can be converted to $\mathbf{E}$ by applications of these rules. This is an equivalence relation between wffs. A wff $\mathbf{D}$ is in β-*normal form* iff it has no well-formed parts of the form $[[\lambda \mathbf{x}_\alpha \mathbf{B}_\beta]\mathbf{A}_\alpha]$. Every wff is convertible to one in β-normal form. Indeed, every sequence of contractions (applications of rule 2, combined as necessary with alphabetic changes of bound variables) of a wff is finite; obviously, if such a sequence cannot be extended, it terminates with a wff in β-normal form. (This is called the strong normalization theorem.) By the Church-Rosser Theorem, this wff in β-normal form is unique modulo alphabetic changes of bound variables. For each wff $\mathbf{A}$ we denote by $\downarrow \mathbf{A}$ the first wff (in some enumeration) in β-normal form such that $\mathbf{A}$ conv $\downarrow \mathbf{A}$. Then $\mathbf{D}$ conv $\mathbf{E}$ if and only if $\downarrow \mathbf{D} =\downarrow \mathbf{E}$.

By using the Axiom of Extensionality one can obtain the following derived rule of inference:

> η-*Contraction*. Replace a well-formed part $[\lambda \mathbf{y}_\beta[\mathbf{B}_{\alpha\beta}\mathbf{y}_\beta]]$ of a wff by $\mathbf{B}_{\alpha\beta}$, provided $\mathbf{y}_\beta$ does not occur free in $\mathbf{B}_{\alpha\beta}$.

This rule and its inverse (which is called η-*Expansion*) are sometimes used as additional rules of λ-conversion. See Church 1941, Stenlund 1972, Barendregt 1984, and Barendregt *et al.* 2013 for more information about λ-conversion.

It is worth mentioning (again) that λ-abstraction replaces the need for comprehension axioms in Church's type theory.

3.2 Higher-Order Unification

The challenges in higher-order unification are outlined very briefly. More details on the topic are given in Dowek 2001; its utilization in higher-order theorem provers is also discussed in Benzmüller & Miller 2015.

Definition. A *higher-order unifier* for a pair $\langle \mathbf{A}, \mathbf{B} \rangle$ of wffs is a substitution θ for free occurrences of variables such that $\theta\mathbf{A}$ and $\theta\mathbf{B}$ have the same β-normal form. A higher-order unifier for a set of pairs of wffs is a unifier for each of the pairs in the set.

Higher-order unification differs from first-order unification (Baader & Snyder 2001) in a number of important respects. In particular:

1. Even when a unifier for a pair of wffs exists, there may be no most general unifier (Gould 1966).
2. Higher-order unification is undecidable (Huet 1973b), even in the "second-order" case (Goldfarb 1981).

However, an algorithm has been devised (Huet 1975, Jensen & Pietrzykowski 1976), called *pre-unification*, which will find a unifier for a set of pairs of wffs if one exists. The pre-unifiers computed by Huet's procedure are substitutions that can reduce the original unification problem to one involving only so called *flex-flex* unification pairs. Flex-flex pairs have variable head symbols in both terms to be unified and they are known to always have a solution. The concrete computation of these solutions can thus be postponed or omitted. Pre-unification is utilised in all the resolution based theorem provers mentioned in Section 4.

Pattern unification refers a small subset of unification problems, first studied by Miller 1991, whose identification has been important for the construction of practical systems. In a pattern unification problem every occurrence of an existentially quantified variable is applied to a list of arguments that are all distinct variables bound by either a λ-binder or a universal quantifier in the scope of the existential quantifier. Thus, existentially quantified variables cannot be applied to general terms but a very restricted set of bound variables. Pattern unification, like first-order unification, is decidable and most general unifiers exist for solvable problems. This is why pattern unification is preferrably employed (when applicable) in some state-of-the-art theorem provers for Church's type theory.

3.3 A Unifying Principle

The *Unifying Principle* was introduced in Smullyan 1963 (see also Smullyan 1995) as a tool for deriving a number of basic metatheorems about first-order logic in a uniform way. The principle was extended to elementary type theory by Andrews (1971) and to extensional type theory, that is, Henkin's general semantics without description or choice, by Benzmüller, Brown and Kohlhase (2004). We outline these extensions in some more detail below.

3.3.1 Elementary Type Theory

The Unifying Principle was extended to elementary type theory (the system $\mathcal{T}$ of Section 1.3.2) in Andrews 1971 by applying ideas in Takahashi 1967. This Unifying Principle for $\mathcal{T}$ has been used to establish cut-elimination for $\mathcal{T}$ in Andrews 1971 and completeness proofs for various systems of type theory in Huet 1973a, Kohlhase 1995, and Miller 1983. We first give a definition and then state the principle.

Definition. A property Γ of finite sets of wffs$_o$ is an *abstract consistency property* iff for all finite sets $\mathcal{S}$ of wffs$_o$, the following properties hold (for all wffs **A, B**):

1. If $\Gamma(\mathcal{S})$, then there is no atom $\mathbf{A}$ such that $\mathbf{A} \in \mathcal{S}$ and $[\sim \mathbf{A}] \in \mathcal{S}$.
2. If $\Gamma(\mathcal{S} \cup \{\mathbf{A}\})$, then $\Gamma(\mathcal{S} \cup \downarrow \mathbf{A}\})$.
3. If $\Gamma(\mathcal{S} \cup \{\sim\sim \mathbf{A}\})$, then $\Gamma(\mathcal{S} \cup \{\mathbf{A}\})$.
4. If $\Gamma(\mathcal{S} \cup \{\mathbf{A} \vee \mathbf{B}\})$, then $\Gamma(\mathcal{S} \cup \{\mathbf{A}\})$ or $\Gamma(\mathcal{S} \cup \{\mathbf{B}\})$.
5. If $\Gamma(\mathcal{S} \cup \{\sim [\mathbf{A} \vee \mathbf{B}]\})$, then $\Gamma(\mathcal{S} \cup \{\sim \mathbf{A}, \sim \mathbf{B}\})$.
6. If $\Gamma(\mathcal{S} \cup \{\Pi_{o(o\alpha)}\mathbf{A}_{o\alpha}\})$, then $\Gamma(\mathcal{S} \cup \{\Pi_{o(o\alpha)}\mathbf{A}_{o\alpha}, \mathbf{A}_{o\alpha}\mathbf{B}_\alpha\})$ for each wff $\mathbf{B}_\alpha$.
7. If $\Gamma(\mathcal{S} \cup \{\sim \Pi_{o(o\alpha)}\mathbf{A}_{o\alpha}\})$, then $\Gamma(\mathcal{S} \cup \{\sim \mathbf{A}_{o\alpha}\mathbf{c}_\alpha\})$, for any variable or parameter $\mathbf{c}_\alpha$ which does not occur free in $\mathbf{A}_{o\alpha}$ or any wff in $\mathcal{S}$.

Note that *consistency* is an abstract consistency property.

Unifying Principle for $\mathcal{T}$. If Γ is an abstract consistency property and $\Gamma(\mathcal{S})$, then $\mathcal{S}$ is consistent in $\mathcal{T}$.

Here is a typical application of the Unifying Principle. Suppose there is a procedure $\mathcal{M}$ which can be used to refute sets of sentences, and we wish to show it is complete for $\mathcal{T}$. For any set of sentences, let $\Gamma(\mathcal{S})$ mean that $\mathcal{S}$ is not refutable by $\mathcal{M}$, and show that Γ is an abstract consistency property. Now suppose that $\mathbf{A}$ is a theorem of $\mathcal{T}$. Then $\{\sim \mathbf{A}\}$ is inconsistent in $\mathcal{T}$, so by the Unifying Principle not $\Gamma(\{\sim \mathbf{A}\})$, so $\{\sim \mathbf{A}\}$ is refutable by $\mathcal{M}$.

3.3.2 Extensional Type Theory

Extensions of the above Unifying principle towards Church's type theory with general semantics were studied since the mid nineties. A primary motivation was to support (refutational) completeness investigations for the proof calculi underlying the emerging higher-order automated theorem provers (see Section 4 below). The initial interest was on a fragment of Church's type theory, called *extensional type theory*, that includes the extensionality axioms, but excludes $\iota_{(\alpha(o\alpha))}$ and the axioms for it (description and choice were largely neglected in the automated theorem provers at the time). Analogous to before, a distinction

has been made between extensional type theory with *defined equality* (as in Section 1.2.1, where equality is defined via Leibniz' principle) and extensional type theory with *primitive equality* (e.g. system $\mathcal{Q}_0$ as in Section 1.4 , or, alternatively, a system based on logical constants $\sim_{(oo)}, \vee_{((oo)o)}$, and the $\Pi_{(o(o\alpha))}$'s as in Section 1.2.1, but with additional primitive logical constants $=_{o\alpha\alpha}$ added).

A first attempt towards a Unifying Principle for extensional type theory with primitive equality is presented in Kohlhase 1993. The conditions given there, which are still incomplete[1], were subsequently modified and complemented as follows:

- To obtain a Unifying Principle for extensional type theory with defined equality Benzmüller & Kohlhase 1997 added the following conditions for boolean extensionality, functional extensionality and saturation to the above conditions 1.-7. for $\mathcal{T}$ (their presentation has been adapted; for technical reasons, they also employ a slightly stronger variant for condition 2. based on β-conversion rather than β-normalization):

  8. If $\Gamma(\mathcal{S} \cup \{\mathbf{A}_o = \mathbf{B}_o\})$, then $\Gamma(\mathcal{S} \cup \{\mathbf{A}_o, \mathbf{B}_o\})$ or $\Gamma(\mathcal{S} \cup \{\sim \mathbf{A}_o, \sim \mathbf{B}_o\})$
  9. If $\Gamma(\mathcal{S} \cup \{\mathbf{A}_{\alpha\beta} = \mathbf{B}_{\alpha\beta}\})$, then $\Gamma(\mathcal{S} \cup \{\mathbf{A}_{\alpha\beta}\mathbf{c}_\beta = \mathbf{B}_{\alpha\beta}\mathbf{c}_\beta\})$ for any parameter $\mathbf{c}_\beta$ which does not occur free in $\mathcal{S}$.
  10. $\Gamma(\mathcal{S} \cup \{\mathbf{A}_o\})$ or $\Gamma(\mathcal{S} \cup \{\sim \mathbf{A}_o\})$

  The saturation condition 10. was required to properly establish the principle. However, since this condition is related to the proof theoretic notion of cut-elimination, it limits the utility of the principle in completeness proofs for machine-oriented calculi. The principle was nevertheless used in Benzmüller & Kohlhase 1998a and Benzmüller 1999a,b to obtain a completeness proof for a system of extensional higher-order resolution. The principle was also applied in Kohlhase 1998 to study completeness for a related extensional higher-order tableau calculus,[2] in which the extensionality rules for Leibniz equality were adapted from Benzmüller & Kohlhase 1998a, respectively Benzmüller 1997.

- Different options for achieving a Unifying Principle for extensional type theory with primitive equality are presented in Benzmüller 1999a (in this work primitive logical constants $=_{o\alpha\alpha}$ were used in addition to $\sim_{(oo)}$, $\vee_{((oo)o)}$, and the $\Pi_{(o(o\alpha))}$'s; such a redundant choice of logical constants is not rare in higher-order theorem provers). One option is to introduce a reflexivity and substitutivity condition. An alternative is to combine a reflexivity condition with a condition connecting primitive with defined equality, so that the substitutivity condition follows. Note that introducing a defined notion of equality based on the Leibniz principle is, of course, still possible in this context (defined equality is denoted in the remainder of this section by $\doteq$ to properly distinguish it from primitive equality $=$):

[1] They e.g. do not entail the saturation and reflexivity properties given below.

[2] The calculus in Kohlhase 1998 is technically still flawed; e.g., to guarantee soundness Skolemization must be used in rule $SIM(fun)$.

8. Not $\Gamma(\mathcal{S} \cup \{\sim [\mathbf{A}_\alpha = \mathbf{A}_\alpha]\})$
9. If $\Gamma(\mathcal{S} \cup \{\mathbf{A}_\alpha = \mathbf{A}_\alpha\})$, then $\Gamma(\mathcal{S} \cup \{\mathbf{A}_\alpha \doteq \mathbf{A}_\alpha\})$
10. $\Gamma(\mathcal{S} \cup \{\mathbf{A}_o\})$ or $\Gamma(\mathcal{S} \cup \{\sim \mathbf{A}_o\})$

The saturation condition 10. still has to be added independent of which option is considered. The principle was applied in Benzmüller 1999 to prove completeness for the extensional higher-order RUE-resolution[3] calculus underlying the higher-order automated theorem prover LEO and its successor LEO-II.

- In Benzmüller *et al.* 2004 the principle is presented in a very general way which allows for various possibilities concerning the treatment of extensionality and equality in the range between elementary type theory and extensional type theory. The principle is applied to obtain completeness proofs for an associated range of natural deduction calculi. The saturation condition is still used in this work.

- Based on insights from Brown's (2004, 2007) thesis, a solution for replacing the undesirable saturation condition by two weaker conditions is presented in Benzmüller *et al.* 2009; this work also further studies the relation between saturation and cut-elimination. The two weaker conditions, termed mating and decomposition, are easier to demonstrate than saturation in completeness proofs for machine-oriented calculi. They are (omitting some type information in the second one and abusing notation):

  1. If $\Gamma(\mathcal{S} \cup \{\sim \mathbf{A}_o, \mathbf{B}_o\})$ for atoms $\mathbf{A}_o$ and $\mathbf{B}_o$, then $\Gamma(\mathcal{S} \cup \{\sim [\mathbf{A}_o \doteq \mathbf{B}_o]\})$
  2. If $\Gamma(\mathcal{S} \cup \{\sim [h\overline{\mathbf{A}^n_{\alpha^n}} \doteq h\overline{\mathbf{B}^n_{\alpha^n}}]\})$ where head symbol $h_{\beta\overline{\alpha^n}}$ is a parameter, then there is an i $(1 \leq i \leq n)$ such that $\Gamma(\mathcal{S} \cup \{\sim [\mathbf{A}^i_{\alpha^i} \doteq \mathbf{B}^i_{\alpha^i}]\})$.

  The modified principle is applied in Benzmüller *et al.* 2009 to show completeness for a sequent calculus for extensional type theory with defined equality.

- A further extended Unifying Principle for extensional type theory with primitive equality is presented and used in Backes & Brown 2011 to prove the completeness of a tableau calculus for type theory which incorporates the axiom of choice.

- A closely related and further simplified principle has also been presented and studied in Steen 2018, where it was applied for showing completeness of the paramodulation calculus (Steen 2018) that is underlying the theorem prover Leo-III (Steen & Benzmüller 2018).

[3] RUE stands for resolution by unification and equality.

3.4 Cut-Elimination anf Cut-Simulation

Cut-elimination proofs (see also the SEP entry on proof theory) for Church's type theory, which are often closely related to such proofs (Takahashi 1967 and 1970, Prawitz 1968, Mints 1999) for other formulations of type theory, may be found in Andrews 1971, Dowek & Werner 2003, and Brown 2004. In Benzmüller *et al.* 2009 it is shown how certain wffs_o, such as axioms of extensionality, descriptions, choice (see Sections 1.3.3 to 1.3.5), and induction, can be used to justify cuts in cut-free sequent calculi for elementary type theory. Moreover, the notions of *cut-simulation* and *cut-strong axioms* are introduced in this work, and the need for omitting defined equality and for eliminating *cut-strong axioms* such as extensionality, description, choice and induction in machine-oriented calculi (e.g. by replacing them with more constrained, goal-directed rules) in order to reduce *cut-simulation* effects are discussed as a major challenge for higher-order automated theorem proving. In other words, including cut-strong axioms in a machine-oriented proof calculus for Church's type theory is essentially as bad as including a cut rule, since the cut rule can be mimicked by them.

3.5 Expansion Proofs

An *expansion proof* is a generalization of the notion of a Herbrand expansion of a theorem of first-order logic; it provides a very elegant, concise, and nonredundant representation of the relationship between the theorem and a tautology which can be obtained from it by appropriate instantiations of quantifiers and which underlies various proofs of the theorem. Miller (1987) proved that a wff $\mathbf{A}$ is a theorem of elementary type theory if and only if $\mathbf{A}$ has an expansion proof.

In Brown 2004 and Brown 2007, this concept is generalized to that of an *extensional expansion proof* to obtain an analogous theorem involving type theory with extensionality.

3.6 The Decision Problem

Since type theory includes first-order logic, it is no surprise that most systems of type theory are undecidable. However, one may look for solvable special cases of the decision problem. For example, the system $\mathcal{Q}_0^1$ obtained by adding to $\mathcal{Q}_0$ the additional axiom $\forall x_\iota \forall y_\iota [x_\iota = y_\iota]$ is decidable.

Although the system $\mathcal{T}$ of elementary type theory is analogous to first-order logic in certain respects, it is a considerably more complex language, and special cases of the decision problem for provability in $\mathcal{T}$ seem rather intractable for the most part. Information about some very special cases of this decision problem may be found in Andrews 1974, and we now summarize this.

A wff of the form $\exists \mathbf{x}^1 \ldots \exists \mathbf{x}^n [\mathbf{A} = \mathbf{B}]$ is a theorem of $\mathcal{T}$ iff there is a substitution θ such that $\theta\mathbf{A}$ conv $\theta\mathbf{B}$. In particular, $\vdash \mathbf{A} = \mathbf{B}$ iff $\mathbf{A}$ conv $\mathbf{B}$, which solves the decision problem for wffs of the form $[\mathbf{A} = \mathbf{B}]$. Naturally, the circumstance that only trivial equality formulas are provable in $\mathcal{T}$ changes drastically when

axioms of extensionality are added to $\mathcal{T}$. $\vdash \exists \mathbf{x}_\beta[\mathbf{A} = \mathbf{B}]$ iff there is a wff $\mathbf{E}_\beta$ such that $\vdash [\lambda \mathbf{x}_\beta[\mathbf{A} = \mathbf{B}]]\mathbf{E}_\beta$, but the decision problem for the class of wffs of the form $\exists \mathbf{x}_\beta[\mathbf{A} = \mathbf{B}]$ is unsolvable.

A wff of the form $\forall \mathbf{x}^1 \ldots \forall \mathbf{x}^n \mathbf{C}$, where $\mathbf{C}$ is quantifier-free, is provable in $\mathcal{T}$ iff $\downarrow \mathbf{C}$ is tautologous. On the other hand, the decision problem for wffs of the form $\exists \mathbf{z} \mathbf{C}$, where $\mathbf{C}$ is quantifier-free, is unsolvable. (By contrast, the corresponding decision problem in first-order logic with function symbols is known to be solvable (Maslov 1967).) Since irrelevant or vacuous quantifiers can always be introduced, this shows that the only solvable classes of wffs of $\mathcal{T}$ in prenex normal form defined solely by the structure of the prefix are those in which no existential quantifiers occur.

4 Automation

4.1 Machine-Oriented Proof Calculi

The development, respectively improvement, of machine-oriented proof calculi for Church's type theory is still a challenge research topic. Compared e.g. to the theoretical and practical maturity achieved in first-order automated theorem proving, the area is still in its infancy. Obviously, the challenges are also much bigger than in first-order logic. The practically way more expressive nature of the term-language of Church's type theory causes a larger, bushier and more difficult to traverse proof search space than in first-order logic. Moreover, remember that unification, which constitutes a very important control and filter mechanism in first-order theorem proving, is undecidable (in general) in type theory; see Section 3.2. On the positive side, however, there is a chance to find significantly shorter proofs than in first-order logic. This is well illustrated with a small, concrete example in Boolos 1987. Clearly, much further progress is needed to further leverage the practical relevance of existing calculi for Church's type theory and their implementations (see Section 4.3). The challenges include

- an appropriate handling of the impredicative nature of Church's type theory (some form of blind guessing cannot generally be avoided in a complete proof procedure, but must be intelligently guided),
- the elimination/reduction of cut-simulation effects (see Section 3.4) caused by defined equality or cut-strong axioms (e.g. extensionality, description, choice, induction) in the search space,
- the general undecidability of unification, rendering it a rather problematic filter mechanism for controlling proof search,
- the invention of suitable heuristics for traversing the search space,
- the provision of suitable term-orderings and their effective exploitation in term rewriting procedures,

- and the development of efficient data structures in combination with strong technical support for essential operations such λ-conversion, substitution and rewriting.

It is planned that future editions of this article further elaborate on machine-oriented proof calculi for Church's type theory. For the time being, however, we provide only a selection of historical and more recent references for the interested reader (see also Section 5 below):

Sequent calculi Schütte 1960, Takahashi 1970, Takeuti 1987, Mints 1999, Brown 2004, 2007, Benzmüller *et al.* 2009

Mating method Andrews 1981, Bibel 1981, Bishop 1999

Resolution calculi Andrews 1971, Huet 1973a, Jensen & Pietrzykowski 1976, Benzmüller 1997, Benzmüller & Kohlhase 1998a, Benzmüller 1999a

Tableau method Kohlhase[4] 1995, 1998, Brown & Smolka 2010, Backes & Brown 2011.

Paramodulation calculi Benzmüller 1999a,b, Steen 2018

4.2 Early Proof Assistants

Early computer systems for proving theorems of Church's type theory (or extensions of it) include HOL (Gordon 1988, Gordon & Melham 1993), TPS (Andrews *et al.* 1996, Andrews & Brown 2006), Isabelle (Paulson 1988a, 1988b), PVS (Owre *et al.* 1996, Shankar 2001), IMPS (Farmer 1993), HOL Light (Harrison 1996), OMEGA (Siekmann *et al.* 2006), and λClam (Richardson *et al.* 1998).

The majority of the above systems focused (at least initially) on interactive proof and provided rather limited support for additional proof automation. Full proof automation was pioneered, in particular, by the TPS project. Progress was made in the nineties, when other projects started similar activities, respectively, enforced theirs. However, the resource investments and achievements were lacking much behind those seen in first-order theorem proving. Significant progress was fostered only later, in particular, through the development of a commonly supported syntax for Church's type theory, called TPTP THF (Sutcliffe & Benzmüller 2010), and the inclusion, from 2009 onwards, of a TPTP THF division in the yearly CASC competitions (kind of world championships for automated theorem proving; see Sutcliffe 2016 for further details).

4.3 Automated Theorem Provers

An selection of theorem provers for Church's type theory is presented. The focus is on systems that have successfully participated in TPTP THF CASC

[4] Different to what is claimed, the presented rules fail to capture an exhaustive extensionality treatment, and so did their implementation in the prover HOT (Konrad 1998); see the respective comment on this in Benzmüller 1997; there are also some soundness issues.

competitions in the past. The latest editions of most mentioned systems can be accessed online via the SystemOnTPTP infrastructure (Sutcliffe 2017). Nearly all mentioned systems produce verifiable proof certificates in the TPTP TSTP syntax. Further details on the automation of Church's type theory are given in Benzmüller & Miller 2014.

The TPS prover (Andrews *et al.* 1996, Andrews & Brown 2006) can be used to prove theorems of elementary type theory or extensional type theory automatically, interactively, or semi-automatically. When searching for a proof automatically, TPS first searches for an expansion proof (Miller 1987) or an extensional expansion proof (Brown 2004, 2007) of the theorem. Part of this process involves searching for acceptable matings (Andrews 1981, Bishop 1999). The behavior of TPS is controlled by sets of flags, also called modes. A simple scheduling mechanism is employed in the latest versions of TPS to sequentially run a about fifty modes for a limited amount of time. TPS was the winner of the first THF CASC competition in 2009.

The LEO-II prover (Benzmüller *et al.* 2015) is the successer of LEO (Benzmüller & Kohlhase 1998b, which was hardwired with the OMEGA proof assistant (LEO stands for Logical Engine of OMEGA). The provers are based on the RUE-resolution calculi developed in Benzmüller 1999. LEO was the first prover to implement calculus rules for extensionality to avoid cut-simulation effects. LEO-II inherits and adapts them, and provides additional calculus rules for description and choice. The prover, which internally collaborates with first-order provers (preferably E) and SAT solvers, has pioneered cooperative higher-order–first-order proof automation. Since the prover is often too weak to find a refutation among the steadily growing set of clauses on its own, some of the clauses in LEO-II's search space attain a special status: they are first-order clauses modulo the application of an appropriate transformation function. Therefore, LEO-II progressively launches time limited calls with these clauses to a first-order theorem prover, and when the first-order prover reports a refutation, LEO-II also terminates. Parts of these ideas were already implemented in the predecessor LEO. Communication between LEO-II and the cooperating first-order theorem provers uses the TPTP language and standards. LEO-II was the winner of the second THF CASC competition in 2010.

The Satallax prover (Brown 2012) is based on a complete ground tableau calculus for Church's type theory with choice (Backes & Brown 2011). An initial tableau branch is formed from the assumptions of a conjecture and negation of its conclusion. From that point on, Satallax tries to determine unsatisfiability or satisfiability of this branch. Satallax progressively generates higher-order formulas and corresponding propositional clauses. Satallax uses the SAT solver MiniSat as an engine to test the current set of propositional clauses for unsatisfiability. If the clauses are unsatisfiable, the original branch is unsatisfiable. Satallax provides calculus rules for extensionality, description and choice. If there are no quantifiers at function types, the generation of higher-order formulas and corresponding clauses may terminate. In that case, if MiniSat reports the final set of clauses as satisfiable, then the original set of higher-order formulas is satisfiable (by a standard model in which all types are interpreted as

finite sets). Satallax was the winner of the THF CASC competition in 2011 and since 2013.

The Isabelle/HOL system (Nipkow *et al.* 2002) has originally been designed as an interactive prover. However, in order to ease user interaction several automatic proof tactics have been added over the years. By appropriately scheduling a subset of these proof tactics, some of which are quite powerful, Isabelle/HOL has since about 2011 been turned also into an automatic theorem prover for TPTP THF (and other TPTP syntax formats), that can be run from a command shell like other provers. The most powerful proof tactics that are scheduled by Isabelle/HOL include the Sledgehammer tool (Blanchette *et al.* 2011), which invokes a sequence of external first-order and higher-order theorem provers, the model finder *Nitpick* (Blanchette & Nipkow 2010), the equational reasoner *simp*, the untyped tableau prover *blast*, the simplifier and classical reasoners *auto*, *force*, and *fast*, and the best-first search procedure *best*. In contrast to all other automated theorem provers mentioned above, the TPTP incarnation of Isabelle/HOL does not yet output proof certificates. Isabelle/HOL was the winner of the THF CASC competition in 2012.

The agsyHOL prover is based on a generic lazy narrowing proof search algorithm. Backtracking is employed and a comparably small search state is maintained. The prover outputs proof terms in sequent style which can be verified in the Agda system.

coqATP implements (the non-inductive) part of the calculus of constructions (Bertot & Casteran 2004). The system outputs proof terms which are accepted as proofs (after the addition of a few definitions) by the Coq proof assistant. The prover employs axioms for functional extensionality, choice, and excluded middle. Boolean extensionality is not supported. In addition to axioms, a small library of basic lemmas is employed.

The Leo-III prover implements a paramodulation calculus for Church's type theory (Steen 2018). The system, which is a descendant of LEO and LEO-II, provides calculus rules for extensionality, description and choice. The system has put an emphasis on the implementation of an efficient set of underlying data structures, on simplification routines and on heuristic rewriting. In the tradition of its predecessors, Leo-III cooperates with first-order reasoning tools using translations to many-sorted first-order logic. The prover accepts every common TPTP syntax dialect and is thus very widely applicable. Recently, the prover has also been extended to natively supports almost every normal higher-order modal logic.

Zipperposition (Bentkamp *et al.* 2018) is new and inspiring higher-order theorem prover which, at the current state of development, is still working for a comparably weak fragment of Church's type theory, called *lambda-free higher-order logic* (a of comprehension-free/-restricted higher-order logic, which is nevertheless supporting λ-notation). The system, which is based on superposition calculi, is developed bottom up, and it is progressively extended towards stronger fragments of Church's type theory and to support other relevant extensions such datatypes, recursive functions and arithmetic.

Various so called *proof hammers*, in the spirit of Isabelle's Sledgehammer

tool, have recently been developed and integrated with modern proof assistants. Prominent examples include HOL(y)Hammer (Kalyszyk & Urban 2015) for HOL Light and a similar hammer (Czaika & Kalyszyk 2018) for the proof assistant Coq.

4.4 (Counter-)Model Finding

Support for finding finite models or countermodels for formulas of Church's type theory was implemented already in the tableau-based prover HOT (Konrad 1998). Restricted (counter-)model finding capabilities are also implemented in the provers Satallax, LEO-II and LEO-III. The most advanced (finite) model finding support is currently realised in the systems Nitpick and Refute. These tools have been integrated with the Isabelle proof assistant. Nitpick is also available as a standalone tool that accepts TPTP THF syntax. The systems are particularly valuable for exposing errors and misconceptions in problem encodings, and for revealing bugs in the THF theorem provers.

5 Applications

5.1 Semantics of Natural Language

Church's type theory plays an important role in the study of the formal semantics of natural language. Pioneering work on this was done by Richard Montague. See his papers "English as a formal language", "Universal grammar", and "The proper treatment of quantification in ordinary English", which are reprinted in Montague 1974. A crucial component of Montague's analysis of natural language is the definition of a tensed intensional logic (Montague 1974, 256), which is an enhancement of Church's type theory. Montague Grammar had a huge impact, and has since been developed in many further directions, not least in Typelogical/Categorical Grammar. Further related work on intensional and higher-order modal logic is presented in Gallin 1975 and Muskens 2006.

5.2 Mathematics and Computer Science

Proof assistants based on Church's Type Theory, including Isabelle/HOL, HOL Light, HOL4, and PVS, have been successfully utilised in a broad range of application in computer science and mathematics.

Applications in computer science include the verification of hardware, software and security protocols. A prominent example is the L4.verified project in which Isabelle/HOL was used to formally prove that the seL4 operating system kernel implements an abstract, mathematical model specifying of what the kernel is supposed to do (Klein *et al.* 2018).

In mathematics proof assistants have been applied for the development of libraries mathematical theories and the verification of challenge theorems. An early example is the mathematical library that was developed since the eighties

in the TPS project. A exemplary list of theorems that were proved automatically with TPS is given in Andrews *et al.* 1996. A very prominent recent example is Hales Flyspeck in which HOL Light was employed to develop a formal proof for Kepler's conjecture (Hales *et al.* 2017). An example that strongly exploits automation support in Isabelle/HOL with Sledgehammer and Nitpick is presented in Benzmüller & Scott 2019. In this work different axiom systems for category theory were explored and compared.

A solid overview on past and ongoing formalisation projects can be obtained by consulting respective sources such as Isabelle's Archive of Formal Proofs, the Journal of Formalized Reasoning, or the THF entries in Sutcliffe's TPTP problem library.

Further improving proof automation within these proof assistants — based on proof hammering tools or on other forms of prover integration — is relevant for minimising interaction effort in future applications.

5.3 Computational Metaphysics and Artificial Intelligence

Church's type theory is a classical logic, but topical applications in philosophy and artificial intelligence often require expressive non-classical logics. In oder to support such applications with reasoning tools for Church's type theory, the shallow semantical embedding technique (see also Section 1.2.2) has been developed that generalizes and extends the ideas underlying the well known standard translation of modal logics to first-order logic. The technique was applied for the assessment of modern variants of the ontological argument with a range of higher-order theorem provers, including LEO-II, Satallax, Nitpick and Isabelle/HOL. In the course of experiments, LEO-II detected an inconsistency in the premises of Gödel's argument, while the provers succeeded in automatically proving Scott's emendation of it and to confirm the consistency of the emended premises. More details on this work are presented in a related SEP article on automated reasoning (see Section 6 on Logic and Philosophy). The semantical embedding approach has been adapted and further extended for a range of other non-classical logics and related applications. In philosophy this includes the encoding and formal assessment of ambitious ethical and metaphysical theories, and in artificial intelligence this includes the mechanisation of deontic logics and normative reasoning as well as an automatic proof of the muddy children puzzle, which is a famous puzzle in epistemic reasoning, respectively dynamic epistemic reasoning.

6 Bibliography

Andrews, P., 1963, "A Reduction of the Axioms for the Theory of Propositional Types", *Fundamenta Mathematicae*, 52:345–350.

——, **1971,** "Resolution in Type Theory", *Journal of Symbolic Logic*, 36(3):414–432; reprinted in Siekmann & Wrightson 1983 and in Benzmüller *et al.* 2008.

——, **1972a,** "General Models and Extensionality", *Journal of Symbolic Logic*, 37(2):395–397; reprinted in Benzmüller *et al.* 2008.

——, **1972b,** "General Models, Descriptions, and Choice in Type Theory", *Journal of Symbolic Logic*, 37(2):385–394 reprinted in Benzmüller *et al.* 2008.

——, **1974,** "Provability in Elementary Type Theory", *Zeitschrift für Mathematische Logic und Grundlagen der Mathematik*, 20:411–418.

——, **1981,** "Theorem proving via general matings", *Journal of the ACM*, 28(2):193–214.

——, **2001,** "Classical Type Theory", in Robinson and Voronkov 2001, Volume 2, Chapter 15, pp. 965–1007.

——, **2002,** *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*, Dordrecht: Kluwer Academic Publishers, second edition.

Andrews, P., Bishop, M., Issar, S., Nesmith, D., Pfenning, F., and Xi, H., 1996, "TPS: A Theorem Proving System for Classical Type Theory", *Journal of Automated Reasoning*, 16(3):321–353; reprinted in Benzmüller *et al.* 2008.

Andrews, P., and Brown, C., 2006, "TPS: A Hybrid Automatic-Interactive System for Developing Proofs", *Journal of Applied Logic*, 4(4):367–395.

Baader, F., and Snyder, W., 2001, "Unification theory", in Robinson and Voronkov 2001, Volume 1, Chapter 8, Amsterdam: Elsevier Science, pp. 445–533.

Backes, J., and Brown, C., 2011, "Analytic tableaux for higher-order logic with choice", *Journal of Automated Reasoning*, 47(4):451–479.

Barendregt, H. P., 1984, *The λ-Calculus*, Series: Studies in Logic and the Foundations of Mathematics, Amsterdam: North-Holland.

Barendregt, H., Dekkers, W., and Statman, R., 2013, *Lambda Calculus with Types*, Cambridge University Press.

Bentkamp, A., Blanchette, J. C., Cruanes, S., Waldmann, U., 2018, "Superposition for Lambda-Free Higher-Order Logic", in Galmiche *et al.* 2018, pp. 28–46.

Benzmüller, C., 1997, *A Calculus and a System Architecture for Extensional Higher-Order Resolution* , Technical report 97–198, Department of Mathematical Sciences, Carnegie Mellon University.

——, **1999a,** *Equality and Extensionality in Automated Higher-Order Theorem Proving*, Ph.D. dissertation, Computer Science Department, Universität des Saarlandes.

——, **1999b,** "Extensional Higher-Order Paramodulation and RUE-Resolution", in Ganzinger 1999, pp. 399–413.

——, **2019,** "Universal (Meta-)Logical Reasoning: Recent Successes", *Science of Computer Programming*, 172:48–62.

Benzmüller, C., and Kohlhase, 1998a, "Extensional Higher-Order Resolution", in Kirchner and Kirchner 1998, pp. 56–71.

——, **1998b,** "System Description: LEO — A Higher-Order Theorem Prover", in Kirchner and Kirchner 1998, pp. 139–143.

Benzmüller, C., Brown, C., and Kohlhase, M., 2004, "Higher-Order Semantics and Extensionality", *Journal of Symbolic Logic*, 69(4):1027–1088.

——, **2009,** "Cut-Simulation and Impredicativity", *Logical Methods in Computer Science*, 5(1:6):1–21.

Benzmüller, C., Brown, C., Siekmann, J., and Statman, R. (eds.), 2008, *Reasoning in Simple Type Theory* (Festschrift in Honor of Peter B. Andrews on his 70th Birthday), King's College London: College Publications.

Benzmüller, C., and Kohlhase, M., 1997, "Model Existence for Higher-Order Logic", *SEKI Publications*, ISSN 1437-4447, number SR-97-09, 1997.

Benzmüller, C., and Miller, D., 2014, "Automation of higher-order logic", in *Computational Logic, Handbook of the History of Logic*, Volume 9, Amsterdam: Elsevier, pp. 215–254.

Benzmüller, C., Paulson, L, and Theiss, F., 2015, "The Higher-Order Prover LEO-II", *Journal of Automated Reasoning*, 55(4):389–404.

Benzmüller, C., and Scott, D., 2019, "Automating Free Logic in HOL, with an Experimental Application in Category Theory", *Journal of Automated Reasoning*, https://doi.org/10.1007/s10817-018-09507-7.

Benzmüller, C., and Woltzenlogel-Paleo, B., 2014, "Automating Gödel's Ontological Proof of God's Existence with Higher-order Automated Theorem Provers", in Schaub, T., Friedrich, G., and O'Sullivan, B. (eds.), *ECAI 2014*, Frontiers in Artificial Intelligence and Applications, Volume 263, Amsterdam: IOS Press, pp. 93–98.

Bertot, Y., and Castéran, P., 2004, *Interactive Theorem Proving and Program Development – Coq'Art: The Calculus of Inductive Constructions*, Series: Texts in Theoretical Computer Science, Berlin: Springer-Verlag.

Bibel, W., 1981, "On Matrices with Connections", *Journal of the ACM*, 28(4):633–645

Bishop, M., 1999, "A Breadth-First Strategy for Mating Search", in Ganzinger 1999, pp. 359–373.

Blanchette, J. C., and Nipkow, T., 2010, "Nitpick: A Counterexample Generator for Higher-Order Logic Based on a Relational Model Finder", in Kaufmann, M. and Paulson, L. C. (eds.), *Interactive Theorem Proving* (ITP), Series: Lecture Notes in Computer Science, Volume 6172, Berlin: Springer-Verlag, pp. 131–146.

Blanchette, J. C., Böhme, S., and Paulson, L. C., 2013, "Extending Sledgehammer with SMT Solvers", Journal of Automated Reasoning 51(1):109–128.

Büchi, J. R., 1953, "Investigation of the Equivalence of the Axiom of Choice and Zorn's Lemma from the Viewpoint of the Hierarchy of Types", *Journal of Symbolic Logic*, 18(2):125–135.

Brown, C., 2004, *Set Comprehension in Church's Type Theory*, Ph.D. dissertation, Department of Mathematical Sciences, Carnegie Mellon University.

——, **2007,** *Automated Reasoning in Higher-Order Logic: Set Comprehension and Extensionality in Church's Type Theory*, Studies in Logic: Logic and Cognitive Systems: Volume 10, London: College Publications.

——, **2012,** "Satallax: An Automatic Higher-Order Prover", in Sattler, U., Gramlich, B., and Miller, D. (eds.), *Automated Reasoning* (IJCAR), Dordrecht: Springer-Verlag, 111–117.

Brown, C., and Smolka, G., 2010, "Analytic Tableaux for Simple Type Theory and its First-Order Fragment", *Logical Methods in Computer Science*, 6(2).

Boolos, G., 1987, "A Curious Inference", *Journal of Philosophical Logic*, 16(1):1–12.

Church, A., 1932, "A Set of Postulates for the Foundation of Logic", *Annals of Mathematics*, 33(2):346–366.

——, **1940,** "A Formulation of the Simple Theory of Types", *Journal of Symbolic Logic*, 5(2):56–68; reprinted in Benzmüller *et al.* 2008.

——, **1941,** *The Calculi of Lambda-Conversion.* Series: Annals of Mathematics Studies, Volume 6, Princeton: Princeton University Press.

——, **1956,** *Introduction to Mathematical Logic*, Princeton, N.J.: Princeton University Press.

Czajka, L., and Kaliszyk, C., 2018, "Hammer for Coq: Automation for Dependent Type Theory", *Journal of Automated Reasoning*, 61(1–4):423—453.

Dowek, G., 2001, "Higher-Order Unification and Matching", in Robinson and Voronkov 2001, Volume 2, Chapter 16, Amsterdam: Elsevier Science, pp. 1009–1062.

Dowek, G., and Werner, B., 2003, "Proof Normalization Modulo", *Journal of Symbolic Logic*, 68(4):1289–1316.

Farmer, W. M., Guttmann, J. D., and Thayer, F. J., 1993, "IMPS: An interactive mathematical proof system", *Journal of Automated Reasoning*, 11(2):213—248.

——, **2008,** "Seven Virtues of Simple Type Theory", *Journal of Applied Logic*, 6(3):267–286.

Gallin, D., 1975, "Intensional and Higher-Order Modal Logic", Amsterdam: North-Holland.

Galmiche, D., Schulz, S., and Sebastiani, R. (eds.), 2018 *Automated Reasoning* (IJCAR), Series: Lecture Notes in Computer Science, Volume 10900, Cham: Springer-Verlag, pp. 108–116.

Ganzinger, H. (ed.), 1999, *Automated Deduction – CADE-16*, Series: Lecture Notes in Artifical Intelligence, Volume 1632.

Gödel, K., 1931, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I", *Monatshefte für Mathematik und Physik*,

38(1):173–198. Translated in Gödel 1986, 144–195, and in van Heijenoort 1967, 596–616.

——, **1986,** *Collected Works* (Volume I), Oxford: Oxford University Press.

Goldfarb, W., 1981, "The Undecidability of the Second-Order Unification Problem", *Theoretical Computer Science*, 13(2):225–230.

Gordon, M. J. C., 1986, "Why higher-order logic is a good formalism for specifying and verifying hardware", in Milne, G. J., and Subrahmanyam, P. A. (eds.), *Formal Aspects of VLSI Design*, Amsterdam: North-Holland, pp. 153–177.

——, **1988,** "HOL: A Proof Generating System for Higher-Order Logic", in Birtwistle, G., and Subrahmanyam, P. A. (eds.), *VLSI Specification, Verification, and Synthesis*, Dordrecht: Kluwer Academic Publishers, pp. 73–128.

Gordon, M. J., and Melham, T. F., 1993, *Introduction to HOL: A Theorem-Proving Environment for Higher-Order Logic*, Cambridge: Cambridge University Press.

Gould, W. E., 1966, *A Matching Procedure for ω-order Logic*, Ph.D. dissertation, Mathematics Department, Princeton University.

Hales, T., Adams, M., Bauer, G., Dang, T., Harrison, J., Hoang, L., *et al.*, 2017, "A Formal Proof of the Kepler Conjecture", *Forum of Mathematics*, Pi, 5, E2.

Harrison, J., 1996, "HOL Light: A Tutorial Introduction", in Srivas, M., and Camilleri, A. (eds.), *Formal Methods in Computer-Aided Design* (FMCAD), Series: Lecture Notes in Computer Science, Volume 1166, Berlin: Springer-Verlag, pp 265–269.

Henkin, L., 1950, "Completeness in the Theory of Types", *Journal of Symbolic Logic*, 15(2):81–91; reprinted in Hintikka 1969 and in Benzmüller *et al.* 2008.

——, **1963,** "A Theory of Propositional Types", *Fundamenta Mathematicae*, 52:323–344.

——, **1975,** "Identity as a logical primitive", *Philosophia*, 5(1–2):31–45.

Hilbert, D., 1928, "Die Grundlagen der Mathematik", *Abhandlungen aus dem mathematischen Seminar der Hamburgischen Universität*, 6:65–85; translated in van Heijenoort 1967, pp. 464–479.

Hintikka, J. (ed.), 1969, *The Philosophy of Mathematics*, Oxford: Oxford University Press.

Huet, G. P., 1973a, "A Mechanization of Type Theory", in Walker, D. E., and Norton, L. (eds.), *Proceedings of the Third International Joint Conference on Artificial Intelligence* (Stanford University), Los Altos, CA: William Kaufman, pp. 139–146.

——, **1973b,** "The undecidability of unification in third order Logic", *Information and Control*, 22(3):257–267.

——, **1975,** "A Unification Algorithm for Typed λ-Calculus", *Theoretical Computer Science*, 1(1):27–57.

Jensen, D. C., Pietrzykowski, T., 1976, "Mechanizing ω-order type theory through unification", *Theoretical Computer Science*, 3(2):123–171.

Kaliszyk, C, and Urban, J., 2015, "HOL(y)Hammer: Online ATP Service for HOL Light", *Mathematics in Computer Science*, 9(1):5–22.

Kirchner, C., and Kirchner, H. (eds.), 1998, *Automated Deduction – CADE-15*, Series: *Lecture Notes in Artificial Intelligence*, Volume 1421, London: Springer-Verlag.

Klein, G., Andronick, J., Fernandez, M., Kuz, I., Murray, T., and Heiser, G., 2018, "Formally verified software in the real world", *Commun. ACM* 61(10):68–77.

Kohlhase, M., 1993, "A Unifying Principle for Extensional Higher-Order Logic", Technical Report 93–153, Department of Mathematics, Carnegie Mellon University.

——, **1995,** "Higher-Order Tableaux", in Baumgartner, P., Hähnle, R., and Posegga, J. (eds.), *Theorem Proving with Analytic Tableaux and Related Methods*, Series: Lecture Notes in Artificial Intelligence, Volume 918, Berlin: Springer-Verlag.

——, **1998,** "Higher-Order Automated Theorem Proving", in Bibel, W., and Schmitt, P. (eds.), *Automated Deduction — A Basis for Applications*, Volume 1, Dordrecht: Kluwer, pp. 431–462.

Konrad, K., 1998, "HOT: A Concurrent Automated Theorem Prover Based on Higher-Order Tableaux", in Grundy, J., and Newey, M. (eds.), *Theorem Proving in Higher Order Logics* (TPHOLs), Series: Lecture Notes in Computer Science, Volume 1479, Berlin: Springer-Verlag, pp. 245–261.

Maslov, S. Ju., 1967, "An Inverse Method for Establishing Deducibility of Nonprenex Formulas of Predicate Calculus", *Soviet Mathematics Doklady*, 8(1):16–19.

Miller, D. A., 1983, *Proofs in Higher-Order Logic*, Ph.D. dissertation, Mathematics Department, Carnegie Mellon University.

——, **1987,** "A Compact Representation of Proofs", *Studia Logica*, 46(4):347–370.

——, **1991,** "A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification", *Journal of Logic and Computation* 1(4):497—536.

Mints, G., 1999, "Cut-Elimination for Simple Type Theory with an Axiom of Choice", *Journal of Symbolic Logic*, 64(2):479–485.

Montague, R., 1974, *Formal Philosophy. Selected Papers Of Richard Montague*, edited and with an introduction by Richmond H. Thomason, New Haven: Yale University Press.

Muskens, R., 2006, "Higher Order Modal Logic", in P. Blackburn, J. Van Benthem, and F. Wolter (eds.), *Handbook of Modal Logic*, Elsevier, pp. 621–653.

Muskens, R., 2007, "Intensional Models for the Theory of Types", *Journal of*

Symbolic Logic, 72(1):98–118.

Nipkow, T., Paulson, L., and Wenzel, M., 2002, *Isabelle/HOL – A Proof Assistant for Higher-Order Logic* Series: Lecture Notes in Computer Science, Volume 2283, Berlin: Springer-Verlag.

Owre, S., Rajan, S., Rushby, J.M., Shankar, N., and Srivas, M., 1996, "PVS: Combining Specification, Proof Checking, and Model Checking", in Alur, R., and Henzinger, T. A. (eds.), *Computer-Aided Verification* (CAV), Series: Lecture Notes in Computer Science, Volume 1102, Berlin: Springer-Verlag, pp. 411–414.

Paulson, L., 1988a, "Isabelle: The Next Seven Hundred Theorem Provers", in Lusk, E., and Overbeek, R. (eds.), *9th International Conference on Automated Deduction*, Series: *Lecture Notes in Computer Science*, Volume 310, Berlin: Springer-Verlag, pp. 772–773.

——, **1988b,** "A Formulation of the Simple Theory of Types (for Isabelle)", in Goos, G., and Hartmanis, J., *COLOG-88*, Series: *Lecture Notes in Computer Science*, Volume 417, Springer-Verlag, pp. 246–274.

Prawitz, D., 1968, "Hauptsatz for Higher Order Logic", *Journal of Symbolic Logic*, 33(3):452–457.

Quine, W., 1956, "Unification of Universes in Set Theory", *Journal of Symbolic Logic*, 21(3):267–279.

Richardson, J, Smaill, A., and Green, I., 1998, "System description: Proof planning in higher-order logic with λClam", *Automated Deduction – CADE-15*, in Kirchner and Kirchner 1998, pp. 129–133.

Robinson, A., and Voronkov, A. (eds.), 2001, *Handbook of Automated Reasoning*, Volumes 1 and 2, Amsterdam: Elsevier Science.

Russell, B., 1903, *The Principles of Mathematics*, Cambridge: Cambridge University Press.

——, **1908,** "Mathematical Logic as Based on the Theory of Types", *American Journal of Mathematics*, 30(3):222–262; reprinted in van Heijenoort 1967, pp. 150–182.

Schönfinkel, M., 1924, "Über die Bausteine der mathematischen Logik", *Mathematische Annalen*, 92(3–4):305–316; translated in van Heijenoort 1967, pp. 355–366.

Schütte, K., 1960, "Syntactical and Semantical Properties of Simple Type Theory", *Journal of Symbolic Logic*, 25(4):305–326.

Shankar, N., 2001, "Using Decision Procedures with a Higher-Order Logic", in Boulton, R. J., and Jackson, P. B. (eds.), *Theorem Proving in Higher Order Logics* (TPHOLs), Series: Lecture Notes in Computer Science, Volume 2152, Berlin: Springer-Verlag, pp. 5–26.

Siekmann, J., and Wrightson, G. (eds.), 1983, *Automation of Reasoning* (Classical Papers on Computational Logic 1967–1970: Vol. 2), Berlin: Springer-Verlag.

Siekmann, J., Benzmüller, C., and Autexier, S., 2006, "Computer sup-

ported mathematics with OMEGA", *Journal of Applied Logic*, 4(4):533–559.

Smullyan, R. M., 1963, "A Unifying Principle in Quantification Theory", *Proceedings of the National Academy of Sciences* (U.S.A.), 49(6):828–832.

——, **1995,** *First-Order Logic*, New York: Dover, second corrected edition.

Steen, A., 2018, *Extensional Paramodulation for Higher-Order Logic and its Effective Implementation Leo-III*, Ph.D. dissertation, Series: Dissertations in Artificial Intelligence (DISKI), Volume 345, Berlin: AKA-Verlag (IOS Press).

Steen, A., and Benzmüller, C., 2018, "The Higher-Order Prover Leo-III", in Galmiche *et al.* 2018, pp. 108–116.

Stenlund, S., 1972, *Combinators, λ-Terms and Proof Theory*, Dordrecht: D. Reidel.

Sutcliffe, G., 2016, "The CADE ATP System Competition – CASC", *AI Magazine*, 37(2):99–101.

——, **2017,** "The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0", *Journal of Automated Reasoning*, 59(4):483–502

Sutcliffe, G., and Benzmüller, C., 2010, "Automated Reasoning in Higher-Order Logic using the TPTP THF Infrastructure", *Journal of Formalized Reasoning*, 3(1):1–27.

Takahashi, M., 1967, "A proof of cut-elimination theorem in simple type-theory", *Journal of the Mathematical Society of Japan*, 19(4):399–410.

——, **1970,** "A system of simple type theory of Gentzen style with inference on extensionality, and the cut elimination in it", *Commentarii Mathematici Universitatis Sancti Pauli*, 18(2):129–147.

Takeuti, G., 1987, *Proof Theory*, North-Holland.

Tarski, A., 1923, "Sur le terme primitif de la Logistique", *Fundamenta Mathematicae*, 4:196–200; translated in Tarski 1956, 1–23.

——, **1956,** *Logic, Semantics, Metamathematics*, Oxford: Oxford University Press.

van Heijenoort, J., 1967, *From Frege to Gödel. A Source Book in Mathematical Logic 1879–1931*, Cambridge, MA: Harvard University Press.

Whitehead, A. N., and Russell, B., 1927a, *Principia Mathematica*, Volume 1, Cambridge: Cambridge University Press, second edition.

——, **1927b,** "Incomplete Symbols", in Whitehead & Russell 1927a, 66–84; reprinted in van Heijenoort 1967, 216–223.

Yasuhara, M., 1975, "The Axiom of Choice in Church's Type Theory" (abstract), *Notices of the American Mathematical Society*, 22 (January): A34.

7 Academic Tools

8 Other Internet Resources

- Mathematics Genealogy Project, Mathematics Department, North Dakota State University.

9 Related Entries

artificial intelligence: logic and | axiom of choice | description | epsilon calculus | incompleteness theorem lambda calculus | logic: classical | logic: higher-order | montague semantics | proof theory | quantifiers | reasoning: automated | Russell, Bertrand | typelogical grammar | type theory |

Acknowledgments Portions of this material are adapted from Andrews 2002 and Andrews 2001, with permission from the author and Elsevier. Benzmüller received funding for his contribution from VolkswagenStiftung.