Abstraction and refinement in higher order logic*

Matt Fairtlough, Michael Mendler, and Xiaochun Cheng **

Department of Computer Science, The University of Sheffield,
Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK
{matt,michael}@dcs.shef.ac.uk

Abstract. We develop within higher order logic (HOL) a general and flexible method of abstraction and refinement, which specifically addresses the problem of handling constraints. The method is based on an interpretation of first-order Lax Logic in HOL, which can be seen as a modal extension of deliverables. It provides a new technique for automating reasoning about behavioural constraints. We show how the method can be applied in several different tasks, for example to achieve a formal separation of the logical and timing aspects of hardware design, and to generate systematically timing constraints for a simple sequential device from a formal proof of its abstract behaviour. The method and all proofs in the paper have been implemented in Isabelle as a definitional extension of the HOL logic. We assume the reader is familiar with higher order logic but do not assume detailed knowledge of circuit design.

1 Introduction

In this paper we develop within HOL a general method of abstraction and refinement, and apply it, by way of an instructive example, to the problem of synthesising timing constraints in sequential circuit design. Fig. 1 illustrates our general approach: we view the abstraction process as one of *separation of concerns*, which involves splitting a concrete model or theory into two dimensions which we term the *abstract* and *constraint* dimensions. Along the abstract dimension we *deduce* abstract consequences of our theory, and this process corresponds to traditional abstraction methods in Artificial Intelligence as presented,

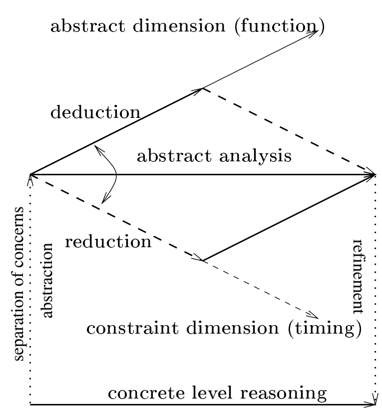

Fig. 1. Diagram of our method

* this work was partially supported by EPSRC under grant GR/L86180
** Department of Computer Science, Reading University. x.cheng@reading.ac.uk

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

for example, in [Pla81]. Along the constraint dimension we track the discrepancies between the abstract and concrete models by computing and *reducing* the constraints under which the abstraction is valid: our method is of practical use insofar as this process can be automated for the constraint domain involved. A key ingredient in our method is an algorithm for recombining the two dimensions at any point, a process strongly related to realizability [Tro98] which we term *refinement*. It is this constraint dimension and the associated concept of refinement that appears to be missing in the AI literature. If we perform an abstraction on a concrete model, develop an abstract analysis and then refine the result, we obtain a concrete consequence which could have been obtained by concrete level reasoning alone; however we argue that the abstraction mechanism is so powerful and natural that it will often be well worth the effort involved in applying it. In our main example we apply the method to the generation of timing constraints for an RS latch, addressing an open problem raised in [HD86] and by [Her89]: a formal proof of circuit behaviour is produced (deduction) at an abstract level at which timing details are elided; an interpretation of the proof as a *constraint λ-term* then yields a concrete proof of correctness incorporating the necessary timing constraints (reduction). The key difference to the previous works is that the constraints are systematically synthesised. They do not need to be introduced at the outset by clairvoyant anticipation of the points at which a concrete-level proof might get stuck.

Abstraction in Artificial Intelligence. Abstraction techniques have been extensively applied and studied in Artificial Intelligence. See for example [Pla81, GW92]. In [GW92] Walsh and Guinchiglia present a very general method to analyse the uses of abstraction in a number of different fields such as theorem proving, planning, problem solving and approximate reasoning. They give an informal definition of abstraction as a process which “allows people to consider what is *relevant* and forget a lot of *irrelevant* details which would get in the way of what they are trying to do”; they go on to note that

> … the process of abstraction is related to the process of *separating*, extracting from a representation an “abstract” representation …

It is however important to realize that separation does not just involve extraction; after separating eggs we may wish to use the white as well as the yolk, perhaps by transforming both and then recombining them. In the context of formal design or verification it is misleading to think of abstraction as extracting the relevant details and discarding the rest as irrelevant; rather we view the process as one of separation of concerns: for example, a system can be split into a deductive part, relating to the broad functionality of the system, and an algorithmic part, representing non-functional aspects such as timing properties. The algorithmic information represents the offset between the abstract system model and the concrete one, *i.e.* it represents the *constraints* under which the abstraction is valid [Men93]. Complementing the abstraction process there should therefore be a dual process of *refinement*, in which the areas of concern are recombined and constraints are (re-)generated. In the examples we present in this

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

paper, the result of refinement is a concrete-level statement that the system has the required functionality provided the constraints are satisfied.

There are many other common sense techniques that can be applied in machine-assisted theorem proving, for example: using lemmas to decompose a proof, making definitions and abbreviations, solving a special case and then generalising, generalising to solve a special case, using flexible variables to track dependencies between separate branches of a proof, goal-directed proof, writing out a paper proof first, and so on. Our approach is to simplify the problem domain and the verification goals via sound abstractions, a technique which can also be used in combination with many of the above methods.

Abstraction and constraints in hardware verification. There is a substantial amount of work on the use of abstraction techniques in formal hardware verification. We take [Fou95, FH91] and [Mel93] as representative publications. A fundamental obstacle to the sound use of abstraction techniques is that the associated constraints accumulate in the opposite direction to the proof effort, whether forward or goal-directed proof methods are used. Thus a situation typically arises where it is not known at the outset what exactly needs to be proved. A knock-on effect of the presence of constraints is a loss of compositionality or modularity with respect to refinement. Although pervasive, this problem of non-compositionality has not often been clearly delineated in the literature. A notable exception is [Eve89]. In his discussion of hierarchical design methods, Eveking presents an example which " ... shows that, in the general case, one has to make a *complete reanalysis* of a switch-level network even if the correctness of the fragments was proven".

In [Mel93] Melham takes two approaches to the problem of handling the constraints involved in the abstraction process. The first process he calls "abstraction within models", and involves theorems of the form $\vdash C \supset M \underset{F}{\mathsf{sat}} S$ where M represents the implementation, S the specification and C the constraints under which the implementation satisfies the specification. F is an abstraction function that serves to relate *e.g.* concrete information to its abstract counterpart. Constraints are handled using explicit rules. In our approach, the abstraction is explicit and the associated constraint rules are implicit, being generated from the abstraction itself. This may prove to be more convenient and flexible in applications.

Melham's second abstraction process is called "abstraction between models" and involves a formal proof of a relationship between hardware models which may be seen as saying "provided certain design rules are followed, the abstract model is a valid description of concrete behaviour". The relevant constraints are automatically satisfied for any correct combination of basic components. [Fou95] takes essentially the same approach. The approach has two limitations: firstly the concrete model at least must be represented by a deep embedding in the object logic, and secondly an adequate set of design rules must be discovered and formalised. While this approach can be highly effective, in general it is not possible to find such a set of design rules, or they are deliberately broken to

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

optimise designs. In contrast, our approach does not require a deep embedding of the concrete model or a formalised set of design rules.

Our contribution. Our approach involves maintaining a close connection between abstraction (the deductive dimension) and constraints (the algorithmic dimension). The algorithmic aspect corresponds to the *calculation* of constraints, and in accordance with this principle we record the "irrelevant" information as sets of constraint λ-terms. Let us consider a tiny example to explain our approach. We define three concrete formulas as follows

$$\psi_1 = \forall s.\underline{(s \geq 5)} \supset (P_1\, a\, \underline{s}) \qquad \psi_2 \ = \ \forall s, y.\underline{(s \geq 9 \cdot y)} \supset (P_2\, (f\, y)\, \underline{s})$$
$$\psi_3 = \forall t, y_1, y_2.(\exists s.\underline{(t \geq s + 35)} \wedge (P_1\, y_1\, \underline{s}) \wedge (P_2\, y_2\, \underline{s})) \supset (Q\, (g(y_1, y_2))\, \underline{t}).$$

These formulas can be seen as representing three components linked together: ψ_1 represents a component P_1 which constantly delivers an output a once 5 time units have elapsed, ψ_2 represents a component P_2 which non-deterministically outputs a value $f(y)$ once $9 \cdot y$ time units have elapsed, and ψ_3 represents their connection to a component Q which takes the outputs from P_1 and P_2 and transforms them using function g, producing the result after a delay of 35 time units. In order to separate the functional and timing aspects, we choose to abstract parameters s from P_1 and P_2 and t from Q, treating the formulas $s \geq 5$, $s \geq 9 \cdot y$ and $t \geq s + 35$ as constraints which should also be hidden at the abstract, functional level. We have indicated the fact that certain parameters and formulas are to be abstracted by underlining them. The results of applying the abstraction process to these formulas ψ_i are then

$$\psi_1{}^\sharp = \lambda s.s \geq 5 : \bigcirc_\forall(P_1\, a) \qquad \psi_2{}^\sharp \ = \ \lambda y, s.s \geq 9 \cdot y : \forall y.\bigcirc_\forall(P_2\, (f\, y))$$
$$\psi_3{}^\sharp = \lambda y_1, y_2, z, t.(\pi_1 z = \pi_2 z \wedge t \geq \pi_1 z + 35) :$$
$$\forall y_1, y_2.((P_1\, y_1) \wedge (P_2\, y_2)) \supset \bigcirc_\forall(Q\, (g(y_1, y_2)))$$

Each new expression $\psi_i{}^\sharp$ is of the form $\psi_i{}^{\sharp_1} : \psi_i{}^{\sharp_2}$ where $\psi_i{}^{\sharp_1}$ is a *constraint λ-term* and $\psi_i{}^{\sharp_2}$ is an *abstract formula*. We shall see later that the : constructor and the modal operator $\bigcirc_\forall$ used in these expressions may be *defined* within HOL so that ψ_i is equivalent to $\psi_i{}^\sharp$ for $1 \leq i \leq 3$. Thus, our abstraction method is a definitional extension of HOL. Informally, the occurrence of the modal operator $\bigcirc_\forall$ in the abstract formula $\bigcirc_\forall(P_1\, a)$ can be explained by reading the formula as: *under some constraint $C\, s$ on the hidden parameter s, $P_1\, a\, s$ holds*, formally $\forall s\,.C\, s \supset P_1\, a\, s$. In other words, $\bigcirc_\forall(P_1\, a)$ indicates that $P_1\, a$ is to be *weakened* by some constraint on the implicit constraint parameter s. Later, we shall see that it is convenient to have a dual modal operator $\bigcirc_\exists(P_1\, a)$ to express a *strengthening* constraint $\exists s\, .\, C\, s \wedge P_1\, a\, s$. The simplest form of abstraction occurs when $C\, s$ is an equality constraint $s = d$ for some term d. Then both $\forall s\, .\, s = d \supset P_1\, a\, s$ and $\exists s\, .\, s = d \wedge P_1\, a\, s$ are equivalent to $P_1\, a\, d$. In this case we may simply write $d : P_1\, a$ as an abstraction of $P_1\, a\, d$, *i.e.* the constraint λ-term is the parameter term d itself.

Having performed an abstraction, we will want to use formal reasoning to deduce abstract properties of the system under consideration. It turns out that the

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

higher order nature of the $\bigcirc_\forall$ modality allows us in many instances to confine our reasoning to a first-order (modal) framework. In section 2 we present a set of rules which are derivable within HOL when extended by abstraction and refinement equations (Fig. 4 for interpreting formulas of the form $p : M$. We may use the rules to deduce abstract consequences of our abstract theory $\Psi^\sharp = {\psi_1}^\sharp, {\psi_2}^\sharp, {\psi_3}^\sharp$, for example there is a constraint λ-term p such that

$$\Psi^\sharp \ \vdash \ p : \exists v.\bigcirc_\forall Q(v). \tag{1}$$

The crucial point here is that the derivation of (1) may be obtained by proceeding along two independent and well-separated dimensions of reasoning: some steps may manipulate the abstract formulas on right-hand side of :, which in this case pertain to functional aspects; other steps may focus on the constraint λ-terms on the left-hand side of :, which contain constraint information. While the analysis of constraint terms, which is done by equational reasoning may in certain domains be decidable and automatic, the abstract formulas require proper logical deduction, which in general is interactive and undecidable. The fact that both aspects are clearly separated by the colon : allows us to benefit from this methodological distinction. At any point, however, the two parts of an abstract conclusion $p : \exists v.\bigcirc_\forall Q(v)$ (whether or not it has been obtained in this structured fashion) may be re-combined to give a concrete-level formula. One solution for p is

$$\iota_{g(a,f(a))}(\mathsf{let}_\forall\ z \Leftarrow {\psi_1}^{\sharp_1}\ \mathsf{in}\ \mathsf{let}_\forall\ w \Leftarrow {\psi_2}^{\sharp_1}\ a\ \mathsf{in}\ {\psi_3}^{\sharp_1}\ a\,(f(a))\,(z,w)).$$

Note that p is playing a dual role: on the one hand it can be seen as a proof term witnessing how the abstract conclusion $\exists v.\bigcirc_\forall Q(v)$ has been reached; on the other hand it can be seen as a constraint (realiser) for refining the conclusion into a concrete level consequence of the theory Ψ. This is achieved by applying the equivalences in section 2 which allow us to calculate that $p : \exists v.\bigcirc_\forall Q(v)$ is equivalent to the concrete level formula

$$\forall u. u \geq (\max 5\,(9 \cdot f(a))) + 35 \supset (Q\,(g(a,f(a)))\,u).$$

This states that the value $g(a,f(a))$ must appear on the output of Q after at most $(\max 5\,(9 \cdot f(a))) + 35$ time units.

2 Higher-order framework: technical details

We take as *base logic* a polymorphic classical higher order logic such as is implemented in Isabelle. We also assume a suitably closed subset Φ of formulas of higher order logic to act as constraints. In our general implementation, we allow arbitrary formulas of HOL, while a restricted set of formulas might be used in more specialised settings. These constraints can appear in λ-terms, as we saw earlier, and indeed it is this feature that gives our approach bite. We use the notation $T \vdash_{\mathrm{B}} M$ to express the fact that M is a consequence of formulas T in the base logic and $\Delta \vdash_{\mathrm{B}} t :: \alpha$ to express the fact that the HOL term t has type α in the context Δ of typed variables.

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

Abstraction and refinement. We introduce the notation $p : M$ as a syntactic abbreviation in the base logic, where the first component p is a constraint λ-term and M is an abstract formula. Fig. 2 gives the raw (*i.e.* untyped) syntax of constraint λ-terms p and the syntax of the abstract language to which M must belong. The formula $p : M$ will only be well-formed under certain conditions on

$$
\begin{aligned}
p ::= {} & z \mid * \mid c \mid (p,p) \mid \pi_1(p) \mid \pi_2(p) \mid \\
& \mathsf{case}\ p\ \mathsf{of}\ [\iota_1(z_1) \to p,\ \iota_2(z_2) \to p] \mid \iota_1(p) \mid \iota_2(p) \mid \\
& \lambda z\,.\,p \mid p\,p \mid \mathsf{val}_\forall(p) \mid \mathsf{val}_\exists(p) \mid \mathsf{let}_\forall\ z \Leftarrow p\ \mathsf{in}\ p \mid \mathsf{let}_\exists\ z \Leftarrow p\ \mathsf{in}\ p \\
& \langle p \mid x \rangle \mid \pi_t\,p \mid \iota_t(p) \mid \mathsf{case}\ p\ \mathsf{of}\ [\iota_x(z) \to p] \\
M ::= {} & A \mid \mathit{false} \mid \mathit{true} \mid M \wedge M \mid M \vee M \mid M \supset M \mid \\
& \bigcirc_\forall M \mid \bigcirc_\exists M \mid \forall x.\,M \mid \exists x.\,M
\end{aligned}
$$

x ranges over object variables, z, z_1, z_2 over proof variables, t over well-formed object-level terms and c over constraint formulas in Φ. A is a meta-variable for atomic formulas $R\,t_1 \dots t_n$. Negated formulas $\neg M$ are defined by $M \supset \mathit{false}$.

Fig. 2. Syntax of constraint λ-terms and abstract formulas.

p and M. In order to define when a pair $p : M$ is properly formed, we give a mapping from formulas M of our abstract language into refinement types $|M|$ of higher order logic according to Fig. 3 and require that $\vdash_{\mathrm{B}} p :: |M|$. We refer to $|M|$ as the *refinement type* of M.
Note that the mapping removes any dependency of types on object level terms, *i.e.* $|M| = |M\{\sigma\}|$ for any substitution σ of terms for object level variables of M. Also note that the image of *false* is the same as that of *true*, *viz.* the unit type $\mathbf{1}$. We must choose a non-empty type for each formula as empty types are inconsistent with our base logic. The meaning of a well-formed pair p :

$\lvert P\rvert := \alpha \quad \text{if } P :: \alpha \Rightarrow \mathbb{B}$	$\lvert M_1 \wedge M_2\rvert := \lvert M_1\rvert \times \lvert M_2\rvert$	$\lvert\bigcirc_\exists M\rvert := \lvert M\rvert \Rightarrow \mathbb{B}$
$\lvert \mathit{true}\rvert := \mathbf{1}$	$\lvert M_1 \vee M_2\rvert := \lvert M_1\rvert + \lvert M_2\rvert$	$\lvert\bigcirc_\forall M\rvert := \lvert M\rvert \Rightarrow \mathbb{B}$
$\lvert \mathit{false}\rvert := \mathbf{1}$	$\lvert M_1 \supset M_2\rvert := \lvert M_1\rvert \Rightarrow \lvert M_2\rvert$	$\lvert\forall x :: \alpha.M\rvert := \alpha \Rightarrow \lvert M\rvert$
		$\lvert\exists x :: \alpha.M\rvert := \alpha \times \lvert M\rvert$

Fig. 3. Refinement types of abstract formulas.

M is now given by Fig. 4 by recursion on the structure of M. We can read these equations in either direction: from left to right they are used to refine an abstract proof/formula pair into the concrete base logic by zipping p and

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

M together to produce $(p : M)^\flat$, thereby completely eliminating the colon. We call this process *refinement*. From right to left, they can be used to completely separate a formula M into a constraint term $M^{\sharp_1}$ and an abstract formula $M^{\sharp_2}$. The process of generating the pair $M^\sharp = M^{\sharp_1} : M^{\sharp_2}$ we call *abstraction*. Let us

$$
\begin{aligned}
(p : \mathit{true}) &= \mathit{true} \\
(p : \mathit{false}) &= \mathit{false} \\
(p : P) &= P\,p \quad \text{if } P :: \alpha \Rightarrow \mathbb{B} \text{ and } p :: \alpha \\
(p : M \wedge N) &= (\pi_1(p) : M) \wedge (\pi_2(p) : N) \\
(p : M \vee N) &= \mathsf{case}\; p \;\mathsf{of}\; [\iota_1(x) \to (x : M),\ \iota_2(y) \to (y : N)] \\
(p : M \supset N) &= \forall z :: |M|.\,(z : M) \supset (p\,z : N) \\
(p : \forall x :: \alpha.\, M) &= \forall u :: \alpha.\,(p\,u : M\{u/x\}) \\
(p : \exists x :: \alpha.\, M) &= \pi_2(p) : M\{\pi_1(p)/x\} \\
(p : \bigcirc_\exists M) &= \exists z :: |M|.\,p\,z \wedge (z : M) \\
(p : \bigcirc_\forall M) &= \forall z :: |M|.\,p\,z \supset (z : M)
\end{aligned}
$$

Fig. 4. Equations for abstraction and refinement.

first look at ψ_1 from the introduction. We can apply the following sequence of equivalences from Fig. 4 and standard equations of higher order logic to generate the abstraction ${\psi_1}^\sharp$: $\forall s\,.\,(s \geq 5) \supset (P_1\,a\,s) \equiv \forall s\,.\,(s \geq 5) \supset (s : P_1\,a) \equiv \forall s\,.\,(\lambda s\,.\,s \geq 5)\,s \supset (s : P_1\,a) \equiv \lambda s\,.\,s \geq 5 : (\bigcirc_\forall(P_1\,a))$. The next example shows that we can pull an arbitrary constraint formula c into a proof term: $c \equiv c \wedge \mathit{true} \equiv \exists z :: \mathbf{1}\,.\,((\lambda z\,.\,c)\,z \wedge z : \mathit{true}) \equiv \lambda z\,.\,c : \bigcirc_\exists \mathit{true}$. Finally, we can pull arbitrary terms out of disjunctions as follows: $M\,p \vee N\,q \equiv p : M \vee q : N \equiv \exists z\,.\,|M| + |N|\,.\,\mathsf{case}\; z \;\mathsf{of}\; [\iota_1(x_1) \to x_1 = p,\ \iota_2(x_2) \to x_2 = q] \wedge z : (M \vee N) \equiv \lambda z\,.\,\mathsf{case}\; z \;\mathsf{of}\; [\iota_1(x_1) \to x_1 = p,\ \iota_2(x_2) \to x_2 = q] : \bigcirc_\exists(M \vee N)$. In a similar way, we can abstract out of conjunctions, implications and quantifiers. In essence, we can abstract out arbitrary sub-terms or formulas out of first-order syntax, which generalises both parameter abstraction [Pla81] and constraint abstraction [Men93].

Theorem 1 (Conservativity over HOL). *Our definitions and equations for formulas of the form $p : M$ are a conservative extension of HOL.*

In fact, our implementation in Isabelle/HOL provides a purely definitional extension using a non-recursive encoding of a new set of logical operators, from which the equations of Fig. 4 are derived.

Abstract reasoning (Deduction). In Fig. 5 we present a set of rules to be used to progress an abstract analysis along the deduction dimension, driven only by the right hand side of :. These rules are a variant of QLL [FW97] and derivable in the base logic from the equations of Fig. 4. They represent a standard first-order, constructive logic with proof terms extended by two modalities $\bigcirc_\exists$ and $\bigcirc_\forall$ which correspond to two independent strong monads, extrapolating the Curry-Howard correspondence between formulas and types.

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

$$\frac{}{\Gamma, z : M, \Gamma' \vdash z : M}\ \mathcal{I} \qquad \frac{\Gamma, z : M, \Gamma' \vdash q : N}{\Gamma, p : M, \Gamma' \vdash q\{p/x\} : N}\ \mathit{Subst} \quad (p :: |M|)$$

$$\frac{\Gamma \vdash p : M \quad \Gamma \vdash q : N}{\Gamma \vdash (p,q) : M \wedge N}\ \wedge_{\mathcal{I}} \qquad \frac{\Gamma \vdash r : M \wedge N}{\Gamma \vdash \pi_1(r) : M}\ \wedge_{\mathcal{E}} \qquad \frac{\Gamma \vdash r : M \wedge N}{\Gamma \vdash \pi_2(r) : N}\ \wedge_{\mathcal{E}}$$

$$\frac{\Gamma \vdash r : M \vee N \quad \Gamma, y : M \vdash p : K \quad \Gamma, z : N \vdash q : K}{\Gamma \vdash \mathsf{case}\ r\ \mathsf{of}\ [\iota_1(y) \to p,\ \iota_2(z) \to q] : K}\ \vee_{\mathcal{E}}$$

$$\frac{\Gamma \vdash p : M}{\Gamma \vdash \iota_1(p) : M \vee N}\ \vee_{\mathcal{I}} \qquad \frac{\Gamma \vdash q : N}{\Gamma \vdash \iota_2(q) : M \vee N}\ \vee_{\mathcal{I}}$$

$$\frac{\Gamma, z : M \vdash p : N}{\Gamma \vdash \lambda z \,.\, p : M \supset N}\ \supset_{\mathcal{I}} \qquad \frac{\Gamma \vdash p : M \supset N \quad \Gamma \vdash q : M}{\Gamma \vdash p\, q : N}\ \supset_{\mathcal{E}}$$

$$\frac{\Gamma \vdash p : M}{\Gamma \vdash \mathsf{val}_Q(p) : \bigcirc_Q M}\ \bigcirc_{\mathcal{I}} \quad \text{if } Q = \forall \text{ or } Q = \exists$$

$$\frac{\Gamma \vdash p : \bigcirc_Q M \quad \Gamma, z : M \vdash q : \bigcirc_Q N}{\Gamma \vdash \mathsf{let}_Q\ z \Leftarrow p\ \mathsf{in}\ q : \bigcirc_Q N}\ \bigcirc_{\mathcal{E}} \quad \text{if } Q = \forall \text{ or } Q = \exists$$

$$\frac{\Gamma \vdash p : M}{\Gamma \vdash \langle p \mid x \rangle : \forall x.M}\ \forall_{\mathcal{I}} \quad (x \text{ not free in } \Gamma) \qquad \frac{\Gamma \vdash p : \forall x.M}{\Gamma \vdash \pi_t\,(p) : M[t/x]}\ \forall_{\mathcal{E}}$$

$$\frac{\Gamma \vdash p : M[t/x]}{\Gamma \vdash \iota_t(p) : \exists x.M}\ \exists_{\mathcal{I}} \qquad \frac{}{\Gamma \vdash * : \mathit{true}}\ \mathit{true}_{\mathcal{I}}$$

$$\frac{\Gamma \vdash r : \exists x.M \quad \Gamma, z : M \vdash p : K}{\Gamma \vdash \mathsf{case}\ r\ \mathsf{of}\ [\iota_x(z) \to p] : K}\ \exists_{\mathcal{E}} \quad (x \text{ not free in } K \text{ or } \Gamma)$$

Fig. 5. Natural Deduction Rules for abstract logic.

Constraint reasoning (Reduction). Constraint reasoning proceeds by equational rewriting of the constraint λ-terms on the left hand side of :. This involves both the standard β, η equations of λ-calculus and special constraint reductions that can be generated from the equations 6. These latter equations in fact provide a computational semantics for the proof term constructors such as $\mathsf{case}\ p_1\ \mathsf{of}\ [\iota_1(z_1) \to p_2,\ \iota_2(z_2) \to p_3]$ or $\mathsf{val}_\forall(p)$ which are special to our constraint λ-calculus. These equations, called γ-equations in [FMW97], are justified by the definition of $p : M$ as an abbreviation in HOL.

$$\begin{aligned} \langle p \mid x \rangle &= \lambda x \,.\, p & \pi_t\,(p) &= p\, t \\ \iota_t(p) &= (t, p) & \mathsf{case}\ r\ \mathsf{of}\ [\iota_x(z) \to p] &= p\{\pi_1(r)/x, \pi_2(r)/z\} \\ \mathsf{val}_Q(x) &= \lambda y.x = y & (\mathsf{let}_Q\ z \Leftarrow p\ \mathsf{in}\ q) &= \lambda x.\exists z.\, p\, z \wedge q\, x \\ c &= d \quad (c, d \in \Phi,\ \vdash_{\mathrm{B}} c \Longleftrightarrow d) & & \end{aligned}$$

Fig. 6. Interpretation of proof terms.

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

We sum up our method in Fig. 9 at the end of the paper.

3 Our method in practice

We give three examples of practical applications of our method. The first provides an extension of constraint logic programming (CLP), the second is an example of a simple combinational circuit with inductive structure and the third is the latch case study first verified by Herbert in [Her89].
Returning to the example specification of the introduction, we observe that the formulas of Ψ are in fact (equivalent to) Horn clauses. As observed in [FMW97], the standard resolution rules for logic programming can be extended by lax resolution rules which also handle the modality $\bigcirc_\forall$ (and in fact, $\bigcirc_\exists$ also). For example,

$$\frac{\Gamma \vdash p : \bigcirc_Q M \qquad \Gamma \vdash q : \bigcirc_Q N}{\Gamma \vdash \wedge_\bigcirc(p,q) : \bigcirc_Q(M \wedge N)} \wedge_\bigcirc \qquad \frac{\Gamma \vdash p : \bigcirc_Q M \qquad \Gamma \vdash r : M \supset N}{\Gamma \vdash \supset_\bigcirc(r,p) : \bigcirc_Q N} \supset_\bigcirc .$$

The first rule states that a constraint p for M and a constraint q for N can be combined to form a constraint $\wedge_\bigcirc(p,q) = \lambda(w,z) \,.\, p\,w \wedge q\,z$ for $M \wedge N$, while the second that a constraint p for M can be propagated through an implication $r : M \supset N$. The resulting constraint $\supset_\bigcirc(r,p)$ is provably equal to $\lambda z \,.\, \exists m \,.\, p\,m \wedge z = r\,m$. The second rule also has a variant more useful in resolution in which the second premise has the form $\Gamma \vdash r : M \supset \bigcirc_Q N$. In [FMW97] we have shown that the standard semantics for CLP can be faithfully represented by a resolution strategy including lax versions of the usual resolution rules. More precisely, the execution of a CLP program corresponds to a lax resolution strategy for proofs and the CLP constraint analysis corresponds to the reduction of proof terms in our framework. We conjecture that our results can be extended to higher order hereditary Harrop formulas. A sensible proof strategy would begin by extending the version of Lambda-Prolog distributed with Isabelle to handle $\bigcirc_\forall$ and $\bigcirc_\exists$.
The second example is taken from [Men93] where Mendler sets himself the task of realising the abstract successor function at the concrete level of bit vectors. He defines $Inc_w :: \mathbb{B}^w \Rightarrow \mathbb{B}^w$ as a cascade of w half-adders which implements the successor function provided the result can be encoded as a word of at most w bits. This overflow constraint is constructed by a recursion mirroring the recursive construction of the incrementor itself. The abstract goal to prove is

$$\forall w, n \,.\, \bigcirc_\forall((\alpha_w \circ Inc_w \circ \rho_w)\, n = n + 1)$$

where $\alpha_w :: \mathbb{B}^w \Rightarrow \mathbb{N}$ and $\rho_w :: \mathbb{N} \Rightarrow \mathbb{B}^w$ are the abstraction and realisation mappings that translate from bit vectors to numbers and conversely. The constraint generated by the lax proof of the goal is $\lambda w, n \,.\, f\ w\ n\ true$ where f is defined recursively by $f\ 0\ n\ c = (n = 0 \wedge \neg c)$ and $f\ (k+1)\ n\ c = f\ k\ (n \div 2)\ (n = 1 \pmod 2 \wedge c)$. This constraint has computational behaviour, but it can also be shown to be equivalent to a flattened version $n + 1 < 2^{w+1}$. The key difference

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

lies in the fact that in certain contexts the recursive version may *evaluate* to the trivial constraint, while the flattened version would have to be *proved* trivial. Note that the inductive basis of the constraint $f\ w\ n\ true$ when $w = 0$, which is $(n = 0 \wedge \neg true)$, is not only unsatisfiable, but in fact inconsistent. In other words, the abstract proof started from an inconsistent base case. Nevertheless, the lax proof returns a satisfiable constraint.

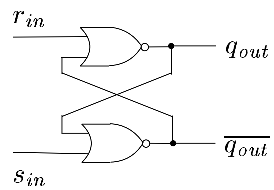

Fig. 7. RS latch

Finally we apply our method to the RS latch, which is illustrated in Fig. 7. The latch is the simplest of three memory devices verified by Herbert in [Her89]. He verifies the device in the HOL system at a detailed timing level, using a discrete model of time and a transport model of basic component behaviour. For example, he defines a NOR gate as $\texttt{NOR2(in0, in1, out, del)} = \forall t.\, \texttt{out}(t + \texttt{del}) = \neg(\texttt{in0}\ t \vee \texttt{in1}\ t)$ and the latch as $\texttt{NOR2(}r_{in}\texttt{,}\ \overline{q_{out}}\texttt{,}\ d_1\texttt{,}\ q_{out}\texttt{)} \wedge \texttt{NOR2(}s_{in}\texttt{,}\ q_{out}\texttt{,}\ d_2\texttt{,}\ \overline{q_{out}}\texttt{)}$. It is clear that the proofs he presents are the result of several iterations which were needed to discover the exact timing constraints necessary to prove them. Similar problems were reported in the work by Hanna and Daeche [HD86]. The main improvement yielded by our approach is that we do not need to make repeated attempts to prove the memory properties, because the timing constraints are synthesised in the course of the abstract analysis.

Our previous work *e.g.* [MF96, Men96] in this area was focussed on combinational devices. The analysis of the system represented by Ψ in the introduction illustrates the basic idea, namely the extraction of a data-dependent timing delay from an abstract proof of functional behaviour. On the other hand, the functional behaviour of sequential devices such as latches depends on timing properties in an essential manner. Thus it is not immediately clear if the same approach will work. We shall however see that it does. The essential idea is to introduce an abstract induction axiom $Ind_P^{\sharp 1} : P \supset (P \supset \bigcirc_\forall P) \supset \bigcirc_\forall P$ which captures the reasoning behind proofs of memory effects. These effects depend on the existence of at least one self-sustaining feedback loop. Given proof terms $p : P$ representing an initial impulse and $q : P \supset \bigcirc_\forall P$ representing the propagation of the impulse through a feedback loop then $Ind_P^{\sharp 1}\, p\, q : \bigcirc_\forall P$ represents the constraints under which the loop is self-sustaining, *i.e.* produces a memory effect.

We verify one of the transitions in fundamental mode, namely the transition that resets the latch: the input r_{in} is held high for a period of time and the output s_{in} is held low for that period and for ever after. Provided that r_{in} is held high for long enough and the intertiality of at least one NOR gate is non-zero, then, after a suitable delay, the output on q_{out} is permanently low. Our specification of the latch and its input excitation, given in Fig. 8, follows Herbert's fairly closely. The inputs r_{in}, s_{in}, and outputs q_{out}, $\overline{q_{out}}$ of the latch are modelled as signals, *i.e.* functions from $\mathbb{N}$ to $\mathbb{B}$. We lift the negation operation $\neg :: \mathbb{B} \Rightarrow \mathbb{B}$ to signals r by defining $\rceil r$ to be $\lambda t :: \mathbb{N}\, .\, \neg(r\, t)$. In his proofs Herbert uses a predicate *During* of type $(\mathbb{N} \Rightarrow \mathbb{B}) \Rightarrow \mathbb{N} \times \mathbb{N} \Rightarrow \mathbb{B}$. For a signal r of type $\mathbb{N} \Rightarrow \mathbb{B}$

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

$$
\begin{aligned}
\theta_1 &= \forall s, t \,.\, (\!| r_{in} |\!)(s,t) \supset (\!| \rceil q_{out} |\!)(s + d_1, t + D_1) \\
\theta_2 &= \forall s_1, t_1, s_2, t_2 \,.\, ((\!| \rceil s_{in} |\!)(s_1, t_1) \wedge (\!| \rceil q_{out} |\!)(s_2, t_2)) \\
&\qquad \supset (\!| \overline{q_{out}} |\!)((\max s_1 s_2) + d_2, (\min t_1 t_2) + D_2) \\
\theta_3 &= \forall s, t \,.\, (\!| \overline{q_{out}} |\!)(s,t) \supset (\!| \rceil q_{out} |\!)(s + d_1, t + D_1) \\
\theta_{p1} &= (\!| r_{in} |\!)(s_a, t_a) \quad \text{and} \quad \theta_{p2} \;=\; \forall t \geq s_a \,.\, (\!| \rceil s_{in} |\!)(s_a, t)
\end{aligned}
$$

Fig. 8. Latch theory.

and pair (s, t) of type $\mathbb{N} \times \mathbb{N}$ the meaning of $\mathit{During}\, r\,(s,t)$ is that r is high (has value *true*) on the whole of the interval $[s, t]$. Thus $\mathit{During}\,\rceil r\,(s,t)$ expresses the fact that r is low (has value *false*) on the interval $[s, t]$. To save space in the presentation of the latch analysis below, we abbreviate $\mathit{During}\, r$ to $(\!| r |\!)$. Fig. 8 gives three base logic axioms specifying the latch itself and two assumptions on the inputs to the latch. The axioms θ_1, θ_2 and θ_3 express the behaviour of the circuit, while θ_{p1} expresses the fact that the input r_{in} is high in the interval $[s_a, t_a]$ and θ_{p2} the fact that the input s_{in} is low in the interval $[s_a, \infty)$. In the formal proof, the parameters s_a, t_a, d_1, d_2, D_1 and D_2 are universally quantified and the assumptions θ_{p1} and θ_{p2} incorporated into the statement of the theorem proved. Currently, the mechanism used is that of Isabelle's locales, which allow a formal development that closely matches our informal presentation.

The main difference to Herbert is that we specify both a delay and an inertiality for each gate. For instance, the clause θ_1 specifies that if the signal r_{in} is high throughout the interval $[s, t]$ then the signal q_{out} is low throughout the interval $[s + d_1, t + D_1]$. The value d_1 represents the maximal delay before the input is reflected in the output, while D_1 represents the minimal length of time the gate continues to propagate the input after it is no longer present. One reason for generalising the system specification in this way is that it is electronically more realistic than the transport model.

Applying the equations of Fig. 4 in reverse and some simple optimisations, we obtain the following abstractions of our circuit theory $\Theta = \{\theta_1, \theta_2, \theta_3, \theta_{p1}, \theta_{p2}\}$:

$$
\begin{aligned}
&{\theta_1}^\sharp = {\theta_1}^{\sharp_1} : (\!| r_{in} |\!) \supset (\!| \rceil q_{out} |\!) && {\theta_1}^{\sharp_1} = \lambda(z_1, z_2) \,.\, (z_1 + d_1, z_2 + D_1) \\
&{\theta_2}^\sharp = {\theta_2}^{\sharp_1} : ((\!| \rceil s_{in} |\!) \wedge (\!| \rceil q_{out} |\!)) \supset (\!| \overline{q_{out}} |\!) && {\theta_2}^{\sharp_1} = \lambda((z_{11}, z_{12}), (z_{21}, z_{22})) \,. \\
& && \qquad ((\max z_{11} z_{21}) + d_2, (\min z_{12} z_{22}) + D_2) \\
&{\theta_3}^\sharp = {\theta_3}^{\sharp_1} : (\!| \overline{q_{out}} |\!) \supset (\!| \rceil q_{out} |\!) && {\theta_3}^{\sharp_1} = \lambda(z_1, z_2) \,.\, (z_1 + d_1, z_2 + D_1) \\
&{\theta_{p1}}^\sharp = {\theta_{p1}}^{\sharp_1} : (\!| r_{in} |\!) && {\theta_{p1}}^{\sharp_1} = (s_a, t_a) \\
&{\theta_{p2}}^\sharp = {\theta_{p2}}^{\sharp_1} : \bigcirc_\forall (\!| \rceil s_{in} |\!) && {\theta_{p2}}^{\sharp_1} = \lambda(s, t) \,.\, s_a \leq s \wedge s \leq t
\end{aligned}
$$

in which the functional and timing aspects have been separated completely from each other.

Induction principles for latching proof. To prove the latching property for the reset transition we will use the following interval induction principle for prop-

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

erties of intervals $I \subseteq \mathbb{N}$: If P holds on an initial interval $I_0 = [b_1, b_2]$ and, whenever $I = [b_1, t_1]$ extends I_0 and P holds on I, then P also holds on an interval properly overlapping I on the right (*i.e.* an interval $[s_2, t_2]$ such that $b_1 \leq s_2 \leq t_1 < t_2$), then P holds on all intervals $[b_1, t]$. This principle is valid for any property P that satisfies $P(I) \wedge P(J) \supset P(I \cup J)$ for all intervals I and J. For convenience we will identify closed intervals $[b_1, b_2]$ with pairs of endpoints (b_1, b_2). As an abstraction of this induction principle we therefore propose the following proof-formula pair:

$$\begin{aligned}
\mathit{Ind}^{\sharp}_P &= (\mathit{Ind}^{\sharp_1}_P : \mathit{Ind}^{\sharp_2}_P) \\
&= (\lambda(b_1, b_2) \,.\, \lambda R \,.\, \lambda(s, t) \,.\, s = b_1 \wedge b_1 \leq t \wedge \mathit{Prog}\, R\, (b_1, b_2) \\
&\qquad : P \supset (P \supset \bigcirc_{\forall} P) \supset \bigcirc_{\forall} P),
\end{aligned}$$

where $P :: \mathbb{N} \times \mathbb{N} \Rightarrow \mathbb{B}$ is any binary relation on natural numbers and $\mathit{Prog}\, R\, (b_1, b_2)$ means that R is *progressive* on (b_1, b_2), *i.e.* whenever (b_1, t_1) is an interval extending (b_1, b_2) then there is an interval (s_2, t_2) strictly overlapping (b_1, t_1) on the right such that (b_1, t_1) and (s_2, t_2) are related by R:

$$\begin{aligned}
\mathit{Prog}\, R\, (b_1, b_2) = (&\forall t_1 \,.\, t_1 \geq b_2 \supset \\
&\quad (\exists s_2, t_2 \,.\, b_1 \leq s_2 \leq t_1 < t_2 \wedge R\, (b_1, t_1)\, (s_2, t_2))).
\end{aligned}$$

$\mathit{Ind}^{\sharp}_P$ is a sound induction axiom for any property P of the form $(\!|Q|\!)$, which can be seen once we have refined it (using the equations in Fig. 4):

$$\begin{aligned}
\mathit{Ind}^{\sharp}_P = \forall (b_1, b_2) \,.\, ((b_1, b_2) : P) \supset \\
&\forall R \,.\, (\forall (x_1, x_2) \,.\, (x_1, x_2) : P \supset \forall (y_1, y_2) \,.\, R(x_1, x_2)(y_1, y_2) \supset (y_1, y_2) : P) \supset \\
&\quad \forall (s, t) \,.\, (s = b_1 \wedge b_1 \leq t \wedge (\forall t_1 \,.\, t_1 \geq b_2 \\
&\qquad \supset (\exists s_2, t_2 \,.\, b_1 \leq s_2 \leq t_1 < t_2 \wedge R\, (b_1, t_1)\, (s_2, t_2)))) \supset (s, t) : P.
\end{aligned}$$

This formula is in fact equivalent to the interval induction property presented above. To further clarify the meaning of $\mathit{Ind}^{\sharp}_P$ we say P is *invariant* under a binary relation $R :: (\mathbb{N} \times \mathbb{N}) \Rightarrow (\mathbb{N} \times \mathbb{N}) \Rightarrow \mathbb{B}$, written $\mathit{Inv}\, P\, R$, when P and R satisfy $\forall s_1, t_1, s_2, t_2 \,.\, (P(s_1, t_1) \wedge R\, (s_1, t_1)\, (s_2, t_2)) \supset P(s_2, t_2)$. Then $\mathit{Ind}^{\sharp}_P$ may be directly re-formulated as

$$\forall b_1, b_2 \,.\, P(b_1, b_2) \supset \forall R \,.\, \mathit{Inv}\, P\, R \supset \mathit{Prog}\, R\, (b_1, b_2) \supset \forall t \geq b_1 \,.\, P(b_1, t),$$

i.e. if P holds in some initial interval (b_1, b_2), and P is invariant under a relation R that is progressive on (b_1, b_2), then P must hold in the infinite interval (b_1, ∞). Now if $R\, (s_1, t_1)\, (s_2, t_2)$ is chosen to mean that $P(s_1, t_1) \supset P(s_2, t_2)$ then P is trivially invariant under R and the statement $\mathit{Prog}\, R\, (b_1, b_2)$ amounts to the condition that whenever $I = [b_1, t_1]$ extends $[b_1, b_2]$ to the right and P holds on I, then P also holds on an interval properly overlapping I on the right. Thus we see that the interval induction principle for P is a consequence of $\mathit{Ind}^{\sharp}_P$; $\mathit{Ind}^{\sharp}_P$ is a mild generalisation which in our implementation we prove in the form $\forall Q \,.\, \mathit{Ind}^{\sharp}_{(\!|Q|\!)}$.

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

Verifying the latch. Let us use our abstractions to carry through an abstract proof and see how the latching constraint comes in. The strategy is to find a proof-formula pair $p_l : \bigcirc_\forall (\!|\rceil q_{out}|\!)$ which can then be refined to give the timing constraint C under which the output q_{out} is low: $\forall s, t \,.\, C(s,t) \supset (\!|\rceil q_{out}|\!)(s,t)$. We want to find a persistent memory property, that is, one that arises from a (single) self-sustaining feedback loop around the latch. This requirement is satisfied by choosing a single application of the abstract induction rule to $(\!|\rceil q_{out}|\!)$. The application $Ind^\sharp\, (\!|\rceil q_{out}|\!)$ is then $ind : (\!|\rceil q_{out}|\!) \supset ((\!|\rceil q_{out}|\!) \supset \bigcirc_\forall (\!|\rceil q_{out}|\!)) \supset \bigcirc_\forall (\!|\rceil q_{out}|\!)$ where $ind = \lambda(b_1, b_2) \,.\, \lambda R \,.\, \lambda(s,t) \,.\, s = b_1 \wedge b_1 \leq t \wedge \mathit{Prog}\, R\,(b_1, b_2)$. So we find proof terms $(p_1, p_2) : (\!|\rceil q_{out}|\!)$ and $R : (\!|\rceil q_{out}|\!) \supset \bigcirc_\forall (\!|\rceil q_{out}|\!)$ and then apply the abstract rule $\supset_\mathcal{E}$ of Fig. 5 twice; we obtain $p_l : (\!|\rceil q_{out}|\!)$ where p_l is the composition $ind\,(p_1, p_2)\, R$ which reduces to $\lambda(s,t) \,.\, s = p_1 \wedge p_1 \leq t \wedge \mathit{Prog}\, R\,(p_1, p_2)$. Now for the details: in the following we will compose formulas and proof terms at the same time in a forward fashion, so that the constraint corresponding to p_l is computed in an incremental manner. In our Isabelle implementation, in contrast, this is carried out in a goal-oriented fashion. Note, however, in either case the constructions can be done in a single direction, no mix-up of forward and backward steps is forced. To find (p_1, p_2) we start with axiom ${\theta_{p1}}^\sharp = (s_a, t_a) : (\!|r_{in}|\!)$ and compose this with the implication axiom ${\theta_1}^\sharp$ to obtain $(p_1, p_2) = (s_a + d_1, t_a + D_1) : (\!|\rceil q_{out}|\!)$, where we have already performed the β-normalisation of the proof term. This is our base case. For the step case we assume (as an hypothesis) we have a proof $(s_1, t_1) : (\!|\rceil q_{out}|\!)$. We will need to lift this to a proof $\mathsf{val}_\forall(s_1, t_1) = \lambda z \,.\, z = (s_1, t_1) : \bigcirc_\forall (\!|\rceil q_{out}|\!)$ using rule $\bigcirc_\mathcal{I}$. Applying $\wedge_\bigcirc$ to the axiom ${\theta_{p2}}^\sharp$ and $\mathsf{val}_\forall(s_1, t_1) : \bigcirc_\forall (\!|\rceil q_{out}|\!)$ we derive

$$p_1 = (\lambda((s,t),(u,v)) \,.\, s_a \leq s \leq t \wedge u = s_1 \wedge v = t_1) : \bigcirc_\forall ((\!|\rceil s_{in}|\!) \wedge (\!|\rceil q_{out}|\!)) . \quad (2)$$

We may use $\supset_\bigcirc$ to propagate this through the implication ${\theta_2}^\sharp$ to obtain $\supset_\bigcirc(r_2, p_1) : \bigcirc_\forall (\!|\overline{q_{out}}|\!)$ where $r_2 = \lambda((z_{11}, z_{12}), (z_{21}, z_{22})) \,.\, ((\max z_{11} z_{21}) + d_2, (\min z_{12} z_{22}) + D_2)$, which after β-normalisation and a little simplication yields:

$$\begin{aligned} &\lambda(z_1, z_2) \,.\, \exists m_{11}, m_{12} \,.\, s_a \leq m_{11} \leq m_{12} \\ &\quad \wedge z_1 = (\max m_{11} s_1) + d_2 \wedge z_2 = (\min m_{12} t_1) + D_2 : \bigcirc_\forall (\!|\overline{q_{out}}|\!) . \end{aligned} \quad (3)$$

The next step is to feed (3) through the implication ${\theta_3}^\sharp$, and β-normalise:

$$\begin{aligned} &\lambda(z_1, z_2) \,.\, \exists m_{11}, m_{12} \,.\, s_a \leq m_{11} \leq m_{12} \wedge z_1 = (\max m_{11} s_1) + d_2 + d_1 \\ &\wedge z_2 = (\min m_{12} t_1) + D_2 + D_1 \ : \ \bigcirc_\forall (\!|\rceil q_{out}|\!) . \end{aligned} \quad (4)$$

We derived this under the assumption $(s_1, t_1) : (\!|\rceil q_{out}|\!)$, which we now discharge:

$$\begin{aligned} &\lambda(s_1, t_1), (z_1, z_2) \,.\, \exists m_{11}, m_{12} \,.\, s_a \leq m_{11} \leq m_{12} \\ &\quad \wedge z_1 = (\max m_{11} s_1) + d_2 + d_1 \wedge z_2 = (\min m_{12} t_1) + D_2 + D_1 \\ &\qquad\qquad : (\!|\rceil q_{out}|\!) \supset \bigcirc_\forall (\!|\rceil q_{out}|\!) . \end{aligned} \quad (5)$$

We have generated the proof term R for the induction step. Now we can take the induction base $(s_a + d_1, t_a + D_1)$ and step function R as arguments of the

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

induction rule $Ind^\sharp\,(\!|\lceil q_{out}|\!)$ to obtain $\lambda(s,t)\,.\,s = s_a+d_1 \wedge s_a+d_1 \leq t \wedge Prog\; R\,(s_a+d_1, t_a+D_1) : \bigcirc_\forall(\!|\lceil q_{out}|\!)$. Expanding, we obtain

$$\begin{aligned}
&\lambda(s,t)\,.\,s = s_a + d_1 \wedge s_a + d_1 \leq t \wedge \\
&\quad (\forall t_1\,.\,t_1 \geq t_a + D_1 \supset \\
&\qquad (\exists s_2, t_2\,.\,s_a + d_1 \leq s_2 \leq t_1 < t_2 \wedge R\,(s_a + d_1, t_1)\,(s_2, t_2))) \\
&\qquad\quad : \bigcirc_\forall(\!|\lceil q_{out}|\!), \quad \text{where} \qquad (6)\\
&R\,(s_a + d_1, t_1)\,(s_2, t_2) = \exists m_{11}, m_{12}\,.\,s_a \leq m_{11} \leq m_{12} \\
&\wedge s_2 = (\max m_{11} s_a + d_1) + d_2 + d_1 \\
&\wedge t_2 = (\min m_{12} t_1) + D_2 + D_1.
\end{aligned}$$

Again, we have β-normalised to keep the expressions as simple as possible. It is now time again to do some constraint reductions. The two equations for s_2 and t_2 allow us to eliminate the $\exists s_2, t_2$ quantifiers, a computation that would be part of simple constraint analysis, *i.e.* incorporated into constraint reductions.

$$\begin{aligned}
&\lambda(s,t)\,.\,s = s_a + d_1 \wedge s_a + d_1 \leq t \wedge \\
&\quad (\forall t_1\,.\,t_1 \geq t_a + D_1 \supset (\exists m_{11}, m_{12}\,.\,s_a \leq m_{11} \leq m_{12} \\
&\qquad s_a + d_1 \leq (\max m_{11} s_a) + d_1 + d_2 + d_1 \leq t_1 < (\min m_{12} t_1) + D_2 + D_1)) \\
&\qquad\quad : \bigcirc_\forall(\!|\lceil q_{out}|\!). \qquad (7)
\end{aligned}$$

The constraint computation will detect that $t_1 < (\min m_{12} t_1) + D_2 + D_1$ is equivalent to $D_2 + D_1 > 0$; that $\max m_{11} s_a$ is the same as m_{11} and hence $(\max m_{11} s_a) + d_1 + d_2 + d_1 \leq t_1$ is equivalent to $m_{11} + 2 \cdot d_1 + d_2 \leq t_1$; and that $s_a + d_1 \leq (\max m_{11} s_a) + d_1 + d_2 + d_1$ is always trivially satisfied. Thus, we end up with

$$\begin{aligned}
&\lambda(s,t)\,.\,s = s_a + d_1 \wedge s_a + d_1 \leq t \wedge \\
&\quad (\forall t_1 \geq t_a + D_1\,.\,\exists m_{11} \geq s_a\,.\,m_{11} + 2d_1 + d_2 \leq t_1 \wedge D_2 + D_1 > 0) \\
&\qquad : \bigcirc_\forall(\!|\lceil q_{out}|\!). \qquad (8)
\end{aligned}$$

At this point, now, it appears we need one slightly more sophisticated argument to deal with the $\forall t_1$ and $\exists m_{11}$ quantifiers: the condition $\forall t_1 \geq t_a + D_1\,.\,\exists m_{11} \geq s_a\,.\,m_{11} + 2d_1 + d_2 \leq t_1$ is logically equivalent to $t_a + D_1 \geq s_a + 2d_1 + d_2$. Given such reasoning is built into constraint reductions, we are looking at the solution form:

$$\begin{aligned}
&\lambda(s,t)\,.\,s = s_a + d_1 \wedge s_a + d_1 \leq t \wedge \\
&\quad t_a + D_1 \geq s_a + 2d_1 + d_2 \wedge D_2 + D_1 > 0 : \bigcirc_\forall(\!|\lceil q_{out}|\!). \qquad (9)
\end{aligned}$$

When (9) is refined back into the base logic we have the desired result:

$$(t_a + D_1 \geq s_a + 2d_1 + d_2 \wedge D_2 + D_1 > 0) \supset \forall t \geq s_a + d_1.\,(\!|\lceil q_{out}|\!)\,(s_a + d_1, t).$$

The predicate $t_a + D_1 \geq s_a + 2d_1 + d_2$ is the external hold constraint: Input r_{in} must remain high for a period of length at least $2d_1 + d_2 - D$ for the latch fully

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

to reset; The second part $D_2 + D_1 > 0$ is the internal memory constraint that at least one of the gates must have non-zero inertia; Finally, the third component $t \geq s_a + d_1$ of the overall constraint states that the propagation delay is d_1.

4 Conclusions

We have presented a conservative extension to higher order logic reasoning that allows for general shallow abstractions and an inverse refinement operation.
The extension uses a computational lambda calculus to represent constraint information that is factored out by the abstraction. The combination between these constraint terms and abstract formulas (syntactically separated by the colon : operator) is a realisability interpretation of constructive logic which formally extends the "deliverables" approach of [McK92], although the motivation and theory of McKinna's work is rather different from ours. It is important to stress that the method does not depend on a constructive (intuitionistic) version of higher-order logic. It is equally applicable to classical HOL. However, the abstraction process applies constructive principles, within HOL.
We believe that this approach provides for an interesting new way of organising proofs in HOL, which allows for the clean and yet sound separation of algorithmic (constraint λ-calculus) and non-algorithmic reasoning (abstract formulas). Abstraction by realisability separation should also open up new avenues for developing heuristic techniques for proof search in HOL, borrowing ideas from AI and constraint programming (CLP). The method yields a natural embedding and generalisation of CLP within HOL. To support our claims we have applied the technique here to the verification of a memory device and demonstrated how abstract functional verification can be combined with constraint synthesis, in one and the same derivation. This solves a methodological problem in the verification of memory devices as highlighted in the work of Hanna and Daeche [HD86] or Herbert [Her89]. Our new approach avoids some of the constraint-related problems found in traditional HOL verifications. It combines the goal-directed backward proving of abstract properties with the simultaneous forward-construction of constraints. We are emphatically *not* proposing our approach as a replacement for the many other effective approaches to verifying hardware in HOL, but we do suggest that it could be used alongside these methods and especially where the handling of differing levels of abstraction is involved.
We are currently evaluating an implementation of our method in the Isabelle theorem prover. The reader familiar with Isabelle may have noticed that the connectives in the abstract formulas of Fig. 4 do not have their expected types. For example, if $P :: \alpha \Rightarrow \mathbb{B}$, $Q :: \beta \Rightarrow \mathbb{B}$, $p :: \alpha$ and $q :: \beta$ then $P\,p \wedge Q\,q = (p, q) : P \wedge Q$ which forces $\wedge$ to have type $(\alpha \Rightarrow \mathbb{B}) \Rightarrow (\beta \Rightarrow \mathbb{B}) \Rightarrow (\alpha \times \beta) \Rightarrow \mathbb{B}$ instead of the usual $\mathbb{B} \Rightarrow \mathbb{B} \Rightarrow \mathbb{B}$. In our implementation we have defined a new set of logical connectives $\sqcup, \sqcap, \sqsupset, \ldots$ to connect abstract formulas and used this definition to *derive* the rules of Fig. 4. This is very straightforward to do, *e.g.* we define the abstract version of conjunction by $P \sqcap Q := \lambda(p, q) \,.\, p : P \wedge q : Q$, where $p : P$ can

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

now be simply defined as $P\,p$. We have used our implementation to synthesise and analyse the timing constraints for the latch and the incrementor example. We are now applying our method to the other memory devices considered in [Her89] and have ambitions to explore its application to formal microprocessor design.

References

[Eve89] H. Eveking. Behavioural consistency between register-transfer- and switch-level descriptions. In D. A. Edwards, editor, *Design Methodologies for VLSI and Computer Architecture*. Elsevier Science, B. V., 1989.

[FH91] M. P. Fourman and R. A. Hexsel. Formal synthesis. In G. Birtwistle, editor, *Proceedings of the IV Higher Order Workshop, Banff, 1990*, 1991.

[FMW97] M. Fairtlough, M. Mendler, and M. Walton. First-order Lax Logic as a Framework for CLP. Technical Report MIPS-9714, Passau University, Department of Mathematics and Computer Science, 1997.

[Fou95] M. P. Fourman. Proof and design. Technical Report ECS-LFCS-95-319, Edinburgh University, Department of Computer Science, 1995.

[FW97] M. Fairtlough and M. Walton. Quantified Lax Logic. Technical Report CS-97-11, Sheffield University, Department of Computer Science, 1997.

[GW92] F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 57:323–389, 1992.

[HD86] F. K. Hanna and N. Daeche. Specification and verification using higher order logic: A case study. In G. M. Milne and P. A. Subrahmanyam, editors, *Formal Aspects of VLSI design, Proc. of the 1985 Edinburgh conf. on VLSI*, pages 179–213. North-Holland, 1986.

[Her89] J. Herbert. Formal reasoning about timing and function of basic memory devices. In Dr. Luc Claesen, editor, *IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Design, Volume 2*, pages 668–687. Elsevier Science Publishers, B.V., 1989.

[McK92] J. H. McKinna. *Deliverables: A Categorical Approach to Program Development in Type Theory*. PhD thesis, Edinburgh University, Department of Computer Science, 1992.

[Mel93] T. F. Melham. *Higher Order Logic and Hardware Verification*. Cambridge University Press, 1993.

[Men93] M. Mendler. *A Modal Logic for Handling Behavioural Constraints in Formal Hardware Verification*. PhD thesis, Edinburgh University, Department of Computer Science, ECS-LFCS-93-255, 1993.

[Men96] M. Mendler. Timing refinement of intuitionistic proofs and its application to the timing analysis of combinational circuits. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proc. 5th Int. Workshop on Theorem Proving with Analytic Tableaux and Related Methods*. Springer, 1996.

[MF96] M. Mendler and M. Fairtlough. Ternary simulation: A refinement of binary functions or an abstraction of real-time behaviour? In M. Sheeran and S. Singh, editors, *Proc. 3rd Workshop on Designing Correct Circuits (DCC'96)*. Springer Electronic Workshops in Computing, 1996.

[Pla81] D. A. Plaisted. Theorem proving with abstraction. *Artificial Intelligence*, 16:47–108, 1981.

[Tro98] A. S. Troelstra. Realizability. In S. Buss, editor, *Handbook of Proof Theory*. Elsevier Science B.V., 1998.

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

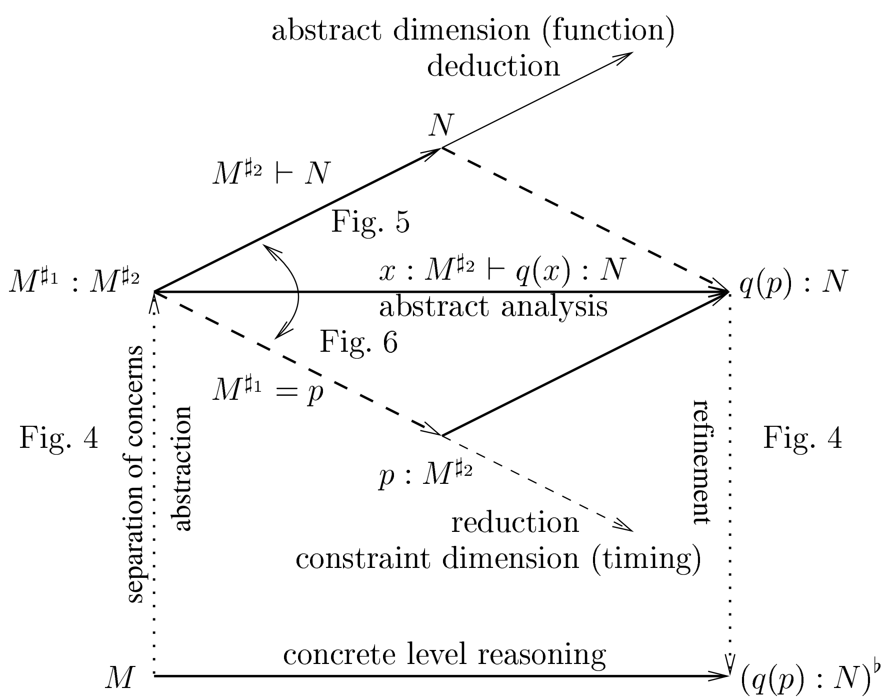

Fig. 9. Summary of our method

M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001