Appendix A: $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$-Formulas and δ-Decidability

We will use a logical language over the real numbers that allows arbitrary *computable real functions* [1]. We write $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$ to represent this language. Intuitively, a real function is computable if it can be numerically simulated up to an arbitrary precision. For the purpose of this paper, it suffices to know that almost all the functions that are needed in describing hybrid systems are Type 2 computable, such as polynomials, exponentiation, logarithm, trigonometric functions, and solution functions of Lipschitz-continuous ordinary differential equations.

More formally, $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}} = \langle \mathcal{F}, > \rangle$ represents the first-order signature over the reals with the set $\mathcal{F}$ of computable real functions, which contains all the functions mentioned above. Note that constants are included as 0-ary functions. $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$-formulas are evaluated in the standard way over the structure $\mathbb{R}_{\mathcal{F}} = \langle \mathbb{R}, \mathcal{F}^{\mathbb{R}}, >^{\mathbb{R}} \rangle$. It is not hard to see that we can put any $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$-formula in a normal form, such that its atomic formulas are of the form $t(x_1, ..., x_n) > 0$ or $t(x_1, ..., x_n) \geq 0$, with $t(x_1, ..., x_n)$ composed of functions in $\mathcal{F}$. To avoid extra preprocessing of formulas, we can explicitly define $\mathcal{L}_{\mathcal{F}}$-formulas as follows.

Definition 1 ($\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$-Formulas). *Let $\mathcal{F}$ be a collection of computable real functions. We define:*

$$t := x \mid f(t(\boldsymbol{x})), \text{ where } f \in \mathcal{F} \text{ (constants are 0-ary functions);}$$
$$\varphi := t(\boldsymbol{x}) > 0 \mid t(\boldsymbol{x}) \geq 0 \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x_i \varphi \mid \forall x_i \varphi.$$

In this setting $\neg\varphi$ is regarded as an inductively defined operation which replaces atomic formulas $t > 0$ with $-t \geq 0$, atomic formulas $t \geq 0$ with $-t > 0$, switches $\wedge$ and $\vee$, and switches $\forall$ and $\exists$.

Definition 2 (Bounded $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$-Sentences). *We define the bounded quantifiers $\exists^{[u,v]}$ and $\forall^{[u,v]}$ as $\exists^{[u,v]}x.\varphi =_{df} \exists x.(u \leq x \wedge x \leq v \wedge \varphi)$ and $\forall^{[u,v]}x.\varphi =_{df} \forall x.((u \leq x \wedge x \leq v) \rightarrow \varphi)$ where u and v denote $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$ terms, whose variables only contain free variables in φ excluding x. A* bounded $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$-sentence *is*

$$Q_1^{[u_1,v_1]} x_1 \cdots Q_n^{[u_n,v_n]} x_n \ \psi(x_1, ..., x_n),$$

where $Q_i^{[u_i,v_i]}$ are bounded quantifiers, and $\psi(x_1, ..., x_n)$ is quantifier-free.

Definition 3 (δ-Variants). *Let $\delta \in \mathbb{Q}^+ \cup \{0\}$, and φ an $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$-formula*

$$\varphi : \ Q_1^{I_1} x_1 \cdots Q_n^{I_n} x_n \ \psi[t_i(\boldsymbol{x}, \boldsymbol{y}) > 0; t_j(\boldsymbol{x}, \boldsymbol{y}) \geq 0],$$

*where $i \in \{1, ...k\}$ and $j \in \{k+1, ..., m\}$. The δ-*weakening *φ^δ of φ is defined as the result of replacing each atom $t_i > 0$ by $t_i > -\delta$ and $t_j \geq 0$ by $t_j \geq -\delta$:*

$$\varphi^\delta : \ Q_1^{I_1} x_1 \cdots Q_n^{I_n} x_n \ \psi[t_i(\boldsymbol{x}, \boldsymbol{y}) > -\delta; t_j(\boldsymbol{x}, \boldsymbol{y}) \geq -\delta].$$

It is clear that $\varphi \rightarrow \varphi^\delta$ (see [2]).

In [3], we have proved that the following δ-decision problem is decidable, which is the basis of our framework.

Theorem 1 (δ-Decidability [3]). *Let $\delta \in \mathbb{Q}^+$ be arbitrary. There is an algorithm which, given any bounded $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-sentence φ, correctly returns one of the following two answers:*

- δ-True*: φ^δ is true.*
- False*: φ is false.*

When the two cases overlap, either answer is correct.

The following theorem states the (relative) complexity of the δ-decision problem. A bounded Σ_n sentence is a bounded $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-sentence with n alternating quantifier blocks starting with $\exists$.

Theorem 2 (Complexity [2]). *Let S be a class of $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-sentences, such that for any φ in S, the terms in φ are in Type 2 complexity class C. Then, for any $\delta \in \mathbb{Q}^+$, the δ-decision problem for bounded Σ_n-sentences in S is in $(\mathsf{\Sigma_n^P})^\mathsf{C}$.*

Basically, the theorem says that increasing the number of quantifier alternations will in general increase the complexity of the problem, unless $\mathsf{P} = \mathsf{NP}$ (recall that $\mathsf{\Sigma_0^P} = \mathsf{P}$ and $\mathsf{\Sigma_1^P} = \mathsf{NP}$). This result can specialized for specific families of functions. For example, with polynomially-computable functions, the δ-decision problem for bounded Σ_n-sentences is $(\mathsf{\Sigma_n^P})$-complete. For more details and results we again point the interested reader to [2].

References

1. Weihrauch, K.: Computable Analysis: An Introduction. Springer (2000)
2. Gao, S., Avigad, J., Clarke, E.M.: Delta-decidability over the reals. In: LICS. (2012) 305–314
3. Gao, S., Avigad, J., Clarke, E.M.: Delta-complete decision procedures for satisfiability over the reals. In: IJCAR. (2012) 286–300

Appendix B: BCF Model in dReach

As an example of dReach's modeling language, we report below the actual dReach file for one of the BCF models (Run#7) analyzed in the paper.

```
#define  EPI_TVP         1.4506
#define  EPI_TV1M       60.0
#define  EPI_TV2M     1150.0
#define  EPI_TWP       200.0
#define  EPI_TW1M       60.0
#define  EPI_TW2M       15.0
#define  EPI_TS1        2.7342
#define  EPI_TS2       16.0
#define  EPI_TFI        0.11
#define  EPI_TO1      400
#define  EPI_TO2        6.0
#define  EPI_TSO1      30.0181
#define  EPI_TSO2       0.9957
#define  EPI_TSI        1.8875
#define  EPI_TWINF      0.07
#define  EPI_THV        0.3
#define  EPI_THVM       0.006
#define  EPI_THVINF     0.006
#define  EPI_THW        0.13
#define  EPI_THWINF     0.006
#define  EPI_THSO       0.13
#define  EPI_THSI       0.13
#define  EPI_THO        0.006
#define  EPI_KWM       65.0
#define  EPI_KS         2.0994
#define  EPI_KSO        2.0458
#define  EPI_UWM        0.03
#define  EPI_US         0.9087
#define  EPI_UO         0.0
#define  EPI_UU         1.55
#define  EPI_USO        0.65
#define  jfi1  0.0
#define  jso1  (u/EPI_TO1)
#define  jsi1  0.0
#define  jfi2  0.0
#define  jso2  (u/EPI_TO2)
#define  jsi2  0.0
#define  jfi3  0.0
#define  jso3  1.0/(EPI_TSO1+((EPI_TSO2- EPI_TSO1)*(1/(1+exp(-2*EPI_KSO*(u- EPI_USO))))))
#define  jsi3  (0 - (w * s)/EPI_TSI)
#define  jfi4  (0 - v * (u - EPI_THV) * (EPI_UU - u)/EPI_TFI)
#define  jso4  (1.0 / (EPI_TSO1+((EPI_TSO2 - EPI_TSO1)*(1/(1+exp(-2*EPI_KSO*(u- EPI_USO)))))))
#define  jsi4  ( 0 - (w * s)/EPI_TSI)
#define  stim  1.0

[0, 2.0] u;
[0, 2.0] v;
[0, 2.0] w;
[0, 2.0] s;
[0, 1] tau;
[0, 1] time;

{mode 1;
invt:    (u >= 0);
         (u <= 0.006);
         (v >= 0);
         (w >= 0);
         (s >= 0);
         (tau >= 0);
flow:
         d/dt[tau] = 1.0;
         d/dt[u] = (stim - jfi1) - (jso1 + jsi1);
```

```
        d/dt[w] = ((1.0 -(u/EPI_TWINF) - w)/(EPI_TW1M + (EPI_TW2M - EPI_TW1M) *
                  (1/(1+exp(-2*EPI_KWM*(u - EPI_UWM))))));
        d/dt[v] = ((1.0 - v)/EPI_TV1M);
        d/dt[s] = (((1/(1+exp( -2 * EPI_KS * (u - EPI_US) ))) - s)/EPI_TS1);
jump:
        (u >= 0.006) ==> @2 (and (tau' = tau) (u' = u) (v'= v) (w' = w) (s' = s));
}

{mode 2;
invt:
        (u >= 0.006);
        (u <= 0.13);
        (v >= 0);
        (w >= 0);
        (s >= 0);
        (tau >= 0);
flow:
        d/dt[tau] = 1.0;
        d/dt[u] = (stim - jfi2) - (jso2 + jsi2);
        d/dt[w] = ((0.94-w)/(EPI_TW1M + (EPI_TW2M - EPI_TW1M) *
                  (1/(1+exp(-2*EPI_KWM*(u - EPI_UWM))))));
        d/dt[v] = (-v/EPI_TV2M);
        d/dt[s] = (((1/(1+exp( -2 * EPI_KS * (u - EPI_US) ))) - s)/EPI_TS1);
jump:
        (u >= 0.13) ==> @3 (and (tau' = tau) (u' = u) (v'= v) (w' = w) (s' = s));
}

{mode 3;
invt:
        (u >= 0.13);
        (u <= 0.3);
        (v >= 0);
        (w >= 0);
        (s >= 0);
        (tau >= 0);
flow:
        d/dt[tau] = 1.0;
        d/dt[u] = (stim - jfi3) - (jso3 + jsi3);
        d/dt[w] = (-w/EPI_TWP);
        d/dt[v] = (-v/EPI_TV2M);
        d/dt[s] = (((1/(1+exp( -2 * EPI_KS * (u - EPI_US) ))) - s)/EPI_TS2);
jump:
        ( u >= 0.3) ==> @4 (and (tau' = tau) (u' = u) (v'= v) (w' = w) (s' = s));
}

{mode 4;
invt:
        (u >= 0.3);
        (v >= 0);
        (w >= 0);
        (s >= 0);
        (tau >= 0);
flow:
        d/dt[tau] = 1.0;
        d/dt[u] =  (stim - jfi4) - (jso4 + jsi4);
        d/dt[w]  = (-w/EPI_TWP);
        d/dt[v]  = (-v/EPI_TVP);
        d/dt[s]  = (((1/(1+exp( -2 * EPI_KS * (u - EPI_US) ))) - s)/EPI_TS2) ;
jump:
        (u > 2.0) ==> @4 (and (tau' = tau) (u' = u) (v'= v) (w' = w) (s' = s));
}

init:  @1 (and (tau = 0) (u = 0.0) (v = 1.0) (w = 1.0) (s = 0.0));

goal:  @4 (and (tau = 1) (u >= 0.3) (u <= 2) (v >= 0) (v <= 2)
           (w >= 0) (w <= 2) (s >= 0) (s <= 2));
```