project

dara

April 10, 2020

Contents

1 Cellular Automata Base

theory *CA-Base*
imports *Main*
begin

datatype *cell* = *Zero* | *One*

fun *flip* :: *cell* ⇒ *cell* **where**
flip One = *Zero* |
flip Zero = *One*

fun *apply-t-times* :: ('*a* ⇒ '*a*) ⇒ '*a* ⇒ *nat* ⇒ '*a* **where**
apply-t-times f a 0 = *a* |
apply-t-times f a (*Suc n*) = *apply-t-times f* (*f a*) *n*

end

2 Elementary Cellular Automata Base

theory *Elementary-CA-Base*
imports *CA-Base*
begin

datatype *neighbourhood* = *Nb cell cell cell*
type-synonym *rule* = *neighbourhood* ⇒ *cell*

fun *apply-nb* :: (*cell* ⇒ *cell*) ⇒ *neighbourhood* ⇒ *neighbourhood* **where**
apply-nb f (*Nb a b c*) = *Nb* (*f a*) (*f b*) (*f c*)

fun *flip-nb* :: *neighbourhood* ⇒ *neighbourhood* **where**
flip-nb nb = *apply-nb flip nb*

fun *sum-nb* :: *neighbourhood* ⇒ *nat* **where**
sum-nb (*Nb a b c*) = *count-list* [*a*, *b*, *c*] *One*

2.1 Basic Rule Properties

fun *mirror* :: *rule* ⇒ *rule* **where**
mirror r (*Nb a b c*)= *r* (*Nb c b a*)

definition *amphichiral* :: *rule* ⇒ *bool* **where**
amphichiral r ≡ *r* = (*mirror r*)

fun *complement* :: *rule* ⇒ *rule* **where**
complement r nb = *flip* (*r* (*flip-nb nb*))

definition *totalistic* :: *rule* ⇒ *bool* **where**
totalistic r ≡ (∀ *nb1 nb2*. *sum-nb nb1* = *sum-nb nb2* ⟶ (*r nb1*) = (*r nb2*))

2.2 Concrete rule examples

definition *null-rule* :: *rule* **where**
null-rule - = *Zero*

fun *r110* :: *rule* **where**
r110 (*Nb One One One*) = *Zero* |
r110 (*Nb One One Zero*) = *One* |
r110 (*Nb One Zero One*) = *One* |
r110 (*Nb One Zero Zero*) = *Zero* |
r110 (*Nb Zero One One*) = *One* |
r110 (*Nb Zero One Zero*) = *One* |
r110 (*Nb Zero Zero One*) = *One* |
r110 (*Nb Zero Zero Zero*) = *Zero*

end

3 Finite Elementary Cellular Automata Base

theory *Finite-Elementary-CA-Base*
imports *Elementary-CA-Base*
begin

3.1 Basis type of all finite elementary 1D CA

type-synonym *state* = *cell list*

datatype *CA* = *CA* (*State* : *state*) (*Rule* : *rule*)

fun *inner-nbhds* :: *state* ⇒ *neighbourhood list* **where**
inner-nbhds (*x*#*y*#*z*#[]) = (*Nb x y z*) # [] |
inner-nbhds (*x*#*y*#*z*#*zs*) = (*Nb x y z*) # (*inner-nbhds* (*y*#*z*#*zs*)) |
inner-nbhds - = []

3.2 Simple properties for CA

definition *width* :: *CA* ⇒ *nat* **where**
width ca = *length* (*State ca*)

definition *wellformed* :: *CA* ⇒ *bool* **where**
wellformed ca ≡ (*width ca* ≥ *3*)

definition *uniform* :: *state* ⇒ *bool* **where**
uniform s ≡ *length* (*remdups s*) = *1*
end

4 Bounded Elementary Cellular Automata

theory *Bounded-Elementary-CA*
imports *Finite-Elementary-CA-Base*
begin

4.1 Basic definitions of Finite Elementary Cellular Automata

fun *nbhds* :: *state* ⇒ *neighbourhood list* **where**
nbhds [] = [] |
nbhds (*x*#[]) = [(*Nb Zero x Zero*)] |
nbhds (*x*#*y*#[]) = [(*Nb Zero x y*), (*Nb x y Zero*)] |
nbhds (*x*#*xs*) = ((*Nb Zero x* (*hd xs*)) # (*inner-nbhds* (*x*#*xs*))) @ [*Nb* (*last* (*butlast xs*)) (*last xs*) *Zero*]

fun *update-CA* :: *CA* ⇒ *CA* **where**
update-CA (*CA s r*) = *CA* (*map r* (*nbhds s*)) *r*

fun *run-t-steps* :: *CA* ⇒ *nat* ⇒ *CA* **where**
run-t-steps ca n = *apply-t-times update-CA ca n*
end

5 Bounded Elementary Cellular Automata Properties

theory *Bounded-Elementary-CA-Properties*
imports *Bounded-Elementary-CA*
begin

definition *stable* :: *CA* ⇒ *bool* **where**
stable ca ≡ *State* (*update-CA ca*) = *State ca*

definition *yields* :: *CA* ⇒ *state* ⇒ *bool* (**infixr** ‹*yields*› *65*) **where**
A yields s ≡ (∃ *n*. *State* (*run-t-steps A n*) = *s* ∧ *n* > *0*)

definition *loops* :: *CA* ⇒ *bool* **where**
loops ca ≡ *ca yields State ca*

fun *garden-of-eden* :: *CA* ⇒ *bool* **where**
garden-of-eden (*CA s r*) = (¬(∃ *s0*. (*CA s0 r*) *yields s*))

fun *reversible* :: *CA* ⇒ *bool* **where**
reversible (*CA - r*) = (∀ *s*. (∃!*s0*. *State* (*update-CA* (*CA s0 r*)) = *s*))

theorem *ca yields State* (*run-t-steps ca 1*)
proof−
  show *?thesis* **using** *yields-def* **by** *blast*
qed

theorem *t1* :*n*>*0* ⟹ *ca yields State* (*run-t-steps ca n*)
  apply(*simp add*: *yields-def*)
  apply(*rule exI*)
  apply(*rule conjI*)
  apply(*auto*)
  done

definition *orphan* :: *state* ⇒ *rule* ⇒ *bool* **where**
orphan s0 r = (∀ *sl sr*. *garden-of-eden* (*CA* (
sl@*s0*@*sr*) *r*))

definition *class1* :: *rule* ⇒ *bool* **where**
class1 r ≡ (∃! *f*. (∀ *s*. (*CA s r*) *yields f* ∧ *uniform f* ∧ *stable* (*CA f r*)))

definition *class2* :: *rule* ⇒ *bool* **where**
class2 r ≡ (∀ *s*. (∃ *f*. (*CA s r*) *yields f* ∧ *loops* (*CA f r*)))

lemma *class1 ca* ⟹ *class2 ca*
  using *class1-def class2-def loops-def* **by** *auto*
end

6 Toroidal Elementary Cellular Automata

theory *Toroidal-Elementary-CA*
  imports *Finite-Elementary-CA-Base*
begin

fun *nbhds* :: *state* ⇒ *neighbourhood list* **where**
nbhds [] = [] |
nbhds (*x*#*xs*) = ((*Nb* (*last xs*) *x* (*hd xs*)) # (*inner-nbhds* (*x*#*xs*))) @ [*Nb* (*last* (*butlast xs*)) (*last xs*) *x*]

fun *update-CA* :: *CA* ⇒ *CA* **where**
update-CA (*CA s r*) = *CA* (*map r* (*nbhds s*)) *r*

fun *run-t-steps* :: *CA* ⇒ *nat* ⇒ *CA* **where**
run-t-steps ca n = *apply-t-times update-CA ca n*

6.1 Basic CA Rule Examples

definition *rule110* :: *CA* **where**
rule110 ≡ *CA* [*Zero*, *One*, *Zero*] *r110*
end

7 Toroidal Elementary Cellular Automata Properties

theory *Toroidal-Elementary-CA-Properties*
  imports *Toroidal-Elementary-CA*
begin

definition *stable* :: *CA* ⇒ *bool* **where**
stable ca ≡ *State* (*update-CA ca*) = *State ca*

definition *yields* :: *CA* ⇒ *state* ⇒ *bool* (**infixr** ‹*yields*› *65*) **where**
A yields s ≡ (∃ *n*. *State* (*run-t-steps A n*) = *s* ∧ *n* > *0*)

definition *loops* :: *CA* ⇒ *bool* **where**
loops ca ≡ *ca yields State ca*

fun *reversible* :: *CA* ⇒ *bool* **where**
reversible (*CA* - *r*) = (∀ *s*. (∃!*s0*. *State* (*update-CA* (*CA s0 r*)) = *s*))

theorem *ca yields State* (*run-t-steps ca 1*)
proof−
  show *?thesis* **using** *yields-def* **by** *blast*
qed

theorem *t1* :*n*>*0* ⟹ *ca yields State* (*run-t-steps ca n*)
  apply(*simp add*: *yields-def*)
  apply(*rule exI*)
  apply(*rule conjI*)
  apply(*auto*)

done

fun *garden-of-eden* :: *CA* ⇒ *bool* **where**
garden-of-eden (*CA s r*) = (¬(∃ *s0*. *State* (*update-CA* (*CA s0 r*)) = *s*))

definition *orphan* :: *state* ⇒ *rule* ⇒ *bool* **where**
orphan s0 r = (∀ *sl sr*. *garden-of-eden* (*CA* (*sl*@*s0*@*sr*) *r*))

lemma *garden-of-eden ca* ⟹ ¬*reversible ca*
apply (*metis garden-of-eden.elims*(*2*) *reversible.simps*)
done

definition *class1* :: *rule* ⇒ *bool* **where**
class1 r ≡ (∃! *f*. (∀ *s*. (*CA s r*) *yields f* ∧ *uniform f* ∧ *stable* (*CA f r*)))

definition *class2* :: *rule* ⇒ *bool* **where**
class2 r ≡ (∀ *s*. (∃ *f*. (*CA s r*) *yields f* ∧ *loops* (*CA f r*)))

lemma *class1 ca* ⟹ *class2 ca*
using *class1-def class2-def loops-def* **by** *auto*
end

8 Infinite Elementary Cellular Automata

theory *Infinite-Elementary-CA*
imports *Elementary-CA-Base*
begin

type-synonym *state* = *int* ⇒ *cell*

datatype *CA* = *CA* (*State* : *state*) (*Rule* : *rule*)

fun *update-state* :: *CA* ⇒ *state* **where**
update-state (*CA s r*) *n* = *r* (*Nb* (*s* (*n*−*1*)) (*s n*) (*s* (*n*+*1*)))

fun *update-CA* :: *CA* ⇒ *CA* **where**
update-CA (*CA s r*) = *CA* (*update-state* (*CA s r*)) *r*

fun *run-t-steps* :: *CA* ⇒ *nat* ⇒ *CA* **where**
run-t-steps ca n = *apply-t-times update-CA ca n*

8.1 Example CAs

definition *state1* :: *state* **where**
state1 s ≡ (*if s* = *0 then One else Zero*)

definition *rule110* :: *CA* **where**
rule110 ≡ *CA state1 r110*

value *State* (*run-t-steps rule110 5*) (−*3*)
end

9 Infinite Elementary Cellular Automata Properties

theory *Infinite-Elementary-CA-Properties*
imports *Infinite-Elementary-CA*
begin

definition *stable* :: *CA* ⇒ *bool* **where**
stable ca ≡ *State* (*update-CA ca*) = *State ca*

definition *uniform* :: *state* ⇒ *bool* **where**
uniform s ≡ ¬ (*surj s*)

definition *yields* :: *CA* ⇒ *state* ⇒ *bool* (**infixr** ‹*yields*› *65*) **where**
A yields s ≡ (∃ *n*. *State* (*run-t-steps A n*) = *s* ∧ *n* > *0*)

definition *loops* :: *CA* ⇒ *bool* **where**
loops ca ≡ *ca yields State ca*

fun *reversible* :: *CA* ⇒ *bool* **where**
reversible (*CA - r*) = (∀ *s*. (∃!*s0*. *State* (*update-CA* (*CA s0 r*)) = *s*))

fun *garden-of-eden* :: *CA* ⇒ *bool* **where**
garden-of-eden (*CA s r*) = (¬(∃ *s0*. *State* (*update-CA* (*CA s0 r*)) = *s*))

lemma *garden-of-eden ca* ⟹ ¬*reversible ca*
by (*metis garden-of-eden.elims*(*2*) *reversible.simps*)

definition *class1* :: *rule* ⇒ *bool* **where**
class1 r ≡ (∃! *f*. (∀ *s*. (*CA s r*) *yields f* ∧ *uniform f* ∧ *stable* (*CA f r*)))

definition *class2* :: *rule* ⇒ *bool* **where**
class2 r ≡ (∀ *s*. (∃ *f*. (*CA s r*) *yields f* ∧ *loops* (*CA f r*)))

lemma *class1 ca* ⟹ *class2 ca*
using *class1-def class2-def loops-def* **by** *auto*

9.1 Concrete examples

definition *zero-state* :: *state* **where**
zero-state - ≡ *Zero*

lemma *uniform zero-state*
  apply(*simp add*: *zero-state-def uniform-def*)
  apply(*auto*)
  done
end

10 2D Cellular Automata Base

theory *TwoDim-CA-Base*
  imports *CA-Base*
begin

datatype *neighbourhood* = *Nb* (*NorthWest*:*cell*) (*North*:*cell*) (*NorthEast*:*cell*)
(*West*:*cell*) (*Centre*:*cell*)(*East*:*cell*)
(*SouthWest*:*cell*) (*South*:*cell*) (*SouthEast*:*cell*)

type-synonym *rule* = *neighbourhood* ⇒ *cell*

fun *apply-nb* :: (*cell* ⇒ *cell*) ⇒ *neighbourhood* ⇒ *neighbourhood* **where**
apply-nb f (*Nb nw n ne w c e sw s se*) = *Nb* (*f nw*) (*f n*) (*f ne*) (*f w*) (*f c*) (*f e*) (*f sw*) (*f s*) (*f se*)

fun *nb-to-list* :: *neighbourhood* ⇒ *cell list* **where**
nb-to-list (*Nb nw n ne w c e sw s se*) = [*nw*, *n*, *ne*, *w*, *c*, *e*, *sw*, *s*, *se*]

fun *list-to-nb* :: *cell list* ⇒ *neighbourhood* **where**
list-to-nb [*nw*, *n*, *ne*, *w*, *c*, *e*, *sw*, *s*, *se*] = *Nb nw n ne w c e sw s se*

fun *flip-nb* :: *neighbourhood* ⇒ *neighbourhood* **where**
flip-nb nb = *apply-nb flip nb*

fun *sum-nb* :: *neighbourhood* ⇒ *nat* **where**
sum-nb nb = *count-list* (*nb-to-list nb*) *One*

fun *complement* :: *rule* ⇒ *rule* **where**
complement r nb = *flip* (*r* (*flip-nb nb*))

definition *totalistic* :: *rule* ⇒ *bool* **where**
totalistic r ≡ (∀ *nb1 nb2*. *sum-nb nb1* = *sum-nb nb2* ⟶ (*r nb1*) = (*r nb2*))

10.1 Game of life

definition *life* :: *rule* **where**
life ca = (*case* (*Centre ca*) *of*
    One ⇒ (*if* (*sum-nb ca*) = *3* ∨ (*sum-nb ca*) = *4*
        then One else Zero) |
    Zero ⇒ (*if* (*sum-nb ca*) = *3*
        then One else Zero))
end

11 Finite 2D Cellular Automata Base

theory *Finite-TwoDim-CA-Base*
  imports *TwoDim-CA-Base*
begin

type-synonym *state* = *cell list list*

datatype *CA* = *CA* (*State* : *state*) (*Rule* : *rule*)

11.1 Simple properties for CA

definition *width* :: *state* ⇒ *nat* **where**
width s ≡ *length s*

definition *height* :: *state* ⇒ *nat* **where**
height s = *length* (*hd s*)

definition *int-width* :: *state* ⇒ *int* **where**
int-width s ≡ *int* (*width s*)

definition *int-height* :: *state* ⇒ *int* **where**
int-height s ≡ *int* (*height s*)

definition *widthCA* :: *CA* ⇒ *nat* **where**
widthCA ca = *width* (*State ca*)

definition *heightCA* :: *CA* ⇒ *nat* **where**
heightCA ca = *height* (*State ca*)

definition *wellformed* :: *state* ⇒ *bool* **where**
wellformed s ≡ (*width s* ≥ *3*) ∧ (*height s* ≥ *3*) ∧ (∀ *i*. *length* (*s* ! (*i mod* (*width s*))) = *height s*)

definition *uniform* :: *state* ⇒ *bool* **where**
uniform s ≡ *length* (*remdups s*) = *1*

definition *oneCentre* :: *state* **where**
oneCentre ≡ [[*Zero*, *Zero*, *Zero*], [*Zero*, *One*, *Zero*], [*Zero*, *Zero*, *Zero*]]

definition *toroidalBlinker* :: *state* **where**
toroidalBlinker ≡
[*replicate 5 Zero*,
[*Zero* ,*Zero*, *One*, *Zero*, *Zero*],
[*Zero*, *Zero*, *One*, *Zero*, *Zero*],
[*Zero*, *Zero*, *One*, *Zero*, *Zero*],
replicate 5 Zero]

definition *boundedBlinker* :: *state* **where**
boundedBlinker ≡ [
[*Zero*, *One*, *Zero*],
[*Zero*, *One*, *Zero*],
[*Zero*, *One*, *Zero*]
]
end

12 Bounded 2D Cellular Automata

theory *Bounded-TwoDim-CA*
imports *Finite-TwoDim-CA-Base*
begin

fun *out-of-bounds* :: *state* ⇒ *int* ⇒ *int* ⇒ *bool* **where**
out-of-bounds s x y = (*if x* ≥ *int-width s* ∨ *x* < *0* ∨ *y* ≥ *int-height s* ∨ *y* < *0 then True else False*)

fun *get-cell* :: *state* ⇒ *int* ⇒ *int* ⇒ *cell* **where**
get-cell s x y = (*if out-of-bounds s x y then Zero else s*!(*nat x*)!(*nat y*))

fun *get-nbhd* :: *state* ⇒ *int* ⇒ *int* ⇒ *neighbourhood* **where**
get-nbhd s x y = *list-to-nb* [*get-cell s* (*x*+*i*) (*y*+*j*). *j* ← *rev* [−*1*..*1*], *i* ← [−*1*..*1*]]

fun *nbhds* :: *state* ⇒ *neighbourhood list list* **where**
nbhds s = (*let h* = (*int-height s*)−*1 in* (*let w* = (*int-width s*)−*1 in*
[[*get-nbhd s x y*. *y* ← [*0*..*h*]]. *x* ← [*0*..*w*]]))

fun *update-CA* :: *CA* ⇒ *CA* **where**
update-CA (*CA s r*) = *CA* (*map* (λ *xs*. *map r xs*) (*nbhds s*)) *r*

fun *run-t-steps* :: *CA* ⇒ *nat* ⇒ *CA* **where**
run-t-steps ca n = *apply-t-times update-CA ca n*
end

13 Bounded 2D Cellular Automata Properties

theory *Bounded-TwoDim-CA-Properties*
  imports *Bounded-TwoDim-CA*
begin

definition *stable* :: *CA* ⇒ *bool* **where**
stable ca ≡ *State* (*update-CA ca*) = *State ca*

definition *uniform* :: *state* ⇒ *bool* **where**
uniform s ≡ *length* (*remdups* (*concat s*)) = *1*

definition *yields* :: *CA* ⇒ *state* ⇒ *bool* (**infixr** ‹*yields*› *65*) **where**
A yields s ≡ (∃ *n*. *State* (*run-t-steps A n*) = *s* ∧ *n* > *0*)

definition *loops* :: *CA* ⇒ *bool* **where**
loops ca ≡ *ca yields State ca*

fun *reversible* :: *CA* ⇒ *bool* **where**
reversible (*CA - r*) = (∀ *s*. (∃!*s0*. *State* (*update-CA* (*CA s0 r*)) = *s*))

fun *garden-of-eden* :: *CA* ⇒ *bool* **where**
garden-of-eden (*CA s r*) = (¬(∃ *s0*. *State* (*update-CA* (*CA s0 r*)) = *s*))

lemma *garden-of-eden ca* ⟹ ¬*reversible ca*
  by (*metis garden-of-eden.elims*(*2*) *reversible.simps*)

definition *class1* :: *rule* ⇒ *bool* **where**
class1 r ≡ (∃! *f*. (∀ *s*. (*CA s r*) *yields f* ∧ *uniform f* ∧ *stable* (*CA f r*)))

definition *class2* :: *rule* ⇒ *bool* **where**
class2 r ≡ (∀ *s*. (∃ *f*. (*CA s r*) *yields f* ∧ *loops* (*CA f r*)))

lemma *class1 ca* ⟹ *class2 ca*
  using *class1-def class2-def loops-def* **by** *auto*
end

14 Toroidal 2D Cellular Automata

theory *Toroidal-TwoDim-CA*
  imports *Finite-TwoDim-CA-Base*
begin

fun *get-cell* :: *state* ⇒ *int* ⇒ *int* ⇒ *cell* **where**
get-cell s x y = *s*!(*nat x*)!(*nat y*)

fun *get-nbhd* :: *state* ⇒ *int* ⇒ *int* ⇒ *neighbourhood* **where**
get-nbhd s x y = (*let w* = *int-width s in* (*let h* = *int-height s in*

list-to-nb [*get-cell s* ((*x*+*i*) *mod w*) ((*y*+*j*) *mod h*). *j* ← *rev* [−*1*..*1*], *i* ← [−*1*..*1*]]))

fun *nbhds* :: *state* ⇒ *neighbourhood list list* **where**
nbhds s = (*let h* = (*int-height s*)−*1 in* (*let w* = (*int-width s*)−*1 in*
[[*get-nbhd s x y*. *y* ← [*0*..*h*]]. *x* ← [*0*..*w*]]))

fun *update-CA* :: *CA* ⇒ *CA* **where**
update-CA (*CA s r*) = *CA* (*map* (λ *xs*. *map r xs*) (*nbhds s*)) *r*

fun *run-t-steps* :: *CA* ⇒ *nat* ⇒ *CA* **where**
run-t-steps ca n = *apply-t-times update-CA ca n*
end

15 Toroidal 2D Cellular Automata Properties

theory *Toroidal-TwoDim-CA-Properties*
imports *Toroidal-TwoDim-CA*
begin

definition *stable* :: *CA* ⇒ *bool* **where**
stable ca ≡ *State* (*update-CA ca*) = *State ca*

definition *uniform* :: *state* ⇒ *bool* **where**
uniform s ≡ *length* (*remdups* (*concat s*)) = *1*

definition *yields* :: *CA* ⇒ *state* ⇒ *bool* (**infixr** ‹*yields*› *65*) **where**
A yields s ≡ (∃ *n*. *State* (*run-t-steps A n*) = *s* ∧ *n* > *0*)

definition *loops* :: *CA* ⇒ *bool* **where**
loops ca ≡ *ca yields State ca*

fun *reversible* :: *CA* ⇒ *bool* **where**
reversible (*CA - r*) = (∀ *s*. (∃!*s0*. *State* (*update-CA* (*CA s0 r*)) = *s*))

fun *garden-of-eden* :: *CA* ⇒ *bool* **where**
garden-of-eden (*CA s r*) = (¬(∃ *s0*. *State* (*update-CA* (*CA s0 r*)) = *s*))

lemma *garden-of-eden ca* ⟹ ¬*reversible ca*
by (*metis garden-of-eden.elims*(*2*) *reversible.simps*)

definition *class1* :: *rule* ⇒ *bool* **where**
class1 r ≡ (∃! *f*. (∀ *s*. (*CA s r*) *yields f* ∧ *uniform f* ∧ *stable* (*CA f r*)))

definition *class2* :: *rule* ⇒ *bool* **where**
class2 r ≡ (∀ *s*. (∃ *f*. (*CA s r*) *yields f* ∧ *loops* (*CA f r*)))

lemma *class1 ca* ⟹ *class2 ca*

using *class1-def class2-def loops-def* **by** *auto*
end

16 Infinite 2D Cellular Automata

theory *Infinite-TwoDim-CA*
imports *TwoDim-CA-Base*
begin

type-synonym *state* = *int* ⇒ *int* ⇒ *cell*

datatype *CA* = *CA* (*State* : *state*) (*Rule* : *rule*)

fun *update-state* :: *CA* ⇒ *state* **where**
update-state (*CA s r*) *x y* = *r* (*Nb* (*s* (*x*−*1*) (*y*+*1*)) (*s x* (*y*+*1*)) (*s* (*x*+*1*) (*y*+*1*))
(*s* (*x*−*1*) *y*) (*s x y*) (*s* (*x*+*1*) *y*)
(*s* (*x*−*1*) (*y*−*1*)) (*s x* (*y*−*1*)) (*s* (*x*+*1*) (*y*−*1*)))

fun *update-CA* :: *CA* ⇒ *CA* **where**
update-CA (*CA s r*) = *CA* (*update-state* (*CA s r*)) *r*

fun *run-t-steps* :: *CA* ⇒ *nat* ⇒ *CA* **where**
run-t-steps ca n = *apply-t-times update-CA ca n*
end

17 Infinite 2D Cellular Automata Properties

theory *Infinite-TwoDim-CA-Properties*
imports *Infinite-TwoDim-CA*
begin

17.1 Properties

definition *stable* :: *CA* ⇒ *bool* **where**
stable ca ≡ *State* (*update-CA ca*) = *State ca*

definition *uniform* :: *state* ⇒ *bool* **where**
uniform s ≡ ¬ (*surj s*)

definition *yields* :: *CA* ⇒ *state* ⇒ *bool* (**infixr** ‹*yields*› *65*) **where**
A yields s ≡ (∃ *n*. *State* (*run-t-steps A n*) = *s* ∧ *n* > *0*)

definition *loops* :: *CA* ⇒ *bool* **where**
loops ca ≡ *ca yields State ca*

fun *reversible* :: *CA* ⇒ *bool* **where**

reversible $(CA - r) = (\forall s.\ (\exists! s0.\ State\ (update\text{-}CA\ (CA\ s0\ r)) = s))$

fun *garden-of-eden* :: *CA* $\Rightarrow$ *bool* **where**
garden-of-eden $(CA\ s\ r) = (\neg(\exists\ s0.\ State\ (update\text{-}CA\ (CA\ s0\ r)) = s))$

lemma *garden-of-eden ca* $\Longrightarrow$ $\neg$*reversible ca*
by (*metis garden-of-eden.elims*(*2*) *reversible.simps*)

definition *class1* :: *rule* $\Rightarrow$ *bool* **where**
class1 r $\equiv (\exists! f.\ (\forall\ s.\ (CA\ s\ r)\ yields\ f \wedge uniform\ f \wedge stable\ (CA\ f\ r)))$

definition *class2* :: *rule* $\Rightarrow$ *bool* **where**
class2 r $\equiv (\forall\ s.\ (\exists\ f.\ (CA\ s\ r)\ yields\ f \wedge loops\ (CA\ f\ r)))$

lemma *class1 ca* $\Longrightarrow$ *class2 ca*
using *class1-def class2-def loops-def* **by** *auto*

17.2 Concrete examples

definition *oneCentre* :: *state* **where**
oneCentre x y $=$ (*if* $x=0 \wedge y=0$ *then One else Zero*)

definition *blinker* :: *state* **where**
blinker x y $=$ (*if* $x=0 \wedge (y=-1 \vee y=0 \vee y=1)$
then One else Zero)

definition *lifeBlinker* :: *CA* **where**
lifeBlinker $\equiv$ *CA blinker life*

value *State* (*run-t-steps lifeBlinker 0*) (*0*) (*0*)

end